Mathematical Foundations of Deep Learning

Sourangshu Ghosh*

***sourangshug@iisc.ac.in**, **Department of Civil Engineering, Indian Institute of Science Bangalore**

Abstract

Deep learning, as a computational paradigm, fundamentally relies on the synergy of functional approximation, optimization theory, and statistical learning. This work presents an extremely rigorous mathematical framework that formalizes deep learning through the lens of measurable function spaces, risk functionals, and approximation theory. We begin by defining the risk functional as a mapping between measurable function spaces, establishing its structure via Frechet differentiability and variational principles. The hypothesis complexity of neural networks is rigorously analyzed using VC-dimension theory for discrete hypotheses and Rademacher complexity for continuous spaces, providing fundamental insights into generalization and overfitting.

A refined proof of the Universal Approximation Theorem is developed using convolution operators and the Stone-Weierstrass theorem, demonstrating how neural networks approximate arbitrary continuous functions on compact domains with quantifiable error bounds. The depth vs. width trade-off is explored through capacity analysis, bounding the expressive power of networks using Fourier analysis and Sobolev embeddings, with rigorous compactness arguments via the Rellich-Kondrachov theorem.

We extend the theoretical framework to training dynamics, analyzing gradient flow and stationary points, the Hessian structure of optimization landscapes, and the Neural Tangent Kernel (NTK) regime. Generalization bounds are established through PAC-Bayes formalism and spectral regularization, connecting information-theoretic insights to neural network stability. The analysis further extends to advanced architectures, including convolutional and recurrent networks, transformers, generative adversarial networks (GANs), and variational autoencoders, emphasizing their function space properties and representational capabilities.

Finally, reinforcement learning is rigorously examined through deep Q-learning and policy optimization, with applications spanning robotics and autonomous systems. The mathematical depth is reinforced by a comprehensive exploration of optimization techniques, covering stochastic gradient descent (SGD), adaptive moment estimation (Adam), and spectral-based regularization methods. The discussion culminates in a deep investigation of function space embeddings, generalization error bounds, and the fundamental limits of deep learning models.

This work bridges deep learning's theoretical underpinnings with modern advancements, offering a mathematically precise and exhaustive exposition that is indispensable for researchers aiming to rigorously understand and extend the frontiers of deep learning theory.

Contents

1 Mathematical Foundations

Deep learning is a computational paradigm for solving high-dimensional function approximation problems. At its core, it relies on the synergy of:

- **Functional Approximation:** Representing complex, non-linear functions using neural networks. Functional approximation is one of the fundamental concepts in deep learning, and it is integral to how deep learning models, particularly neural networks, solve complex problems. In the context of deep learning, functional approximation refers to the ability of neural networks to represent complex, high-dimensional, and non-linear functions that are often difficult or infeasible to model using traditional mathematical techniques.

- **Optimization Theory:** Solving non-convex optimization problems efficiently. Optimization theory plays a central role in deep learning, as the goal of training deep neural networks is essentially to find the optimal set of parameters (weights and biases) that minimize a predefined objective, often called the loss function. This objective typically measures the difference between the network's predictions and the true values. Optimization techniques guide the training process and determine how well a neural network can learn from data.

- **Statistical Learning Theory:** Understanding generalization behavior on unseen data. Statistical Learning Theory (SLT) provides the mathematical foundation for understanding the behavior of machine learning algorithms, including deep learning models. It offers key insights into how models generalize from training data to unseen data, which is critical for ensuring that deep learning models are not only accurate on the training set but also perform well on new, previously unseen data. SLT helps address fundamental challenges such as overfitting, bias-variance tradeoff, and generalization error.

The problem can be formalized as:

$$\text{Find } f_\theta : X \to Y, \text{ such that } \mathbb{E}_{x,y\sim P}\left[\ell(f_\theta(x), y)\right] \text{ is minimized,} \tag{1.1}$$

where X is the input space, Y is the output space, P is the data distribution, $\ell(\cdot,\cdot)$ is a loss function, θ are parameters of the neural network. This task involves the composition of several disciplines, each of which is explored in rigorous detail below.

1.1 Problem Definition: Risk Functional as a Mapping Between Spaces

1.1.1 Measurable Function Spaces

A measurable space is a fundamental construct in measure theory, denoted by $(\mathcal{X}, \Sigma)$, where $\mathcal{X}$ is a non-empty set referred to as the underlying set or the sample space, and Σ is a σ-algebra, a specific collection of subsets of $\mathcal{X}$ that encodes the notion of measurability. The σ-algebra $\Sigma \subseteq 2^{\mathcal{X}}$, the power set of $\mathcal{X}$, satisfies three axioms, each ensuring a critical aspect of closure under set operations. First, Σ is closed under complementation, meaning that for any set $A \in \Sigma$, its complement

$A^c = \mathcal{X} \setminus A$ is also in Σ. This guarantees the ability to define the "non-occurrence" of measurable events in a mathematically consistent way. Second, Σ is closed under countable unions: for any countable collection $\{A_n\}_{n=1}^{\infty} \subseteq \Sigma$, the union $\bigcup_{n=1}^{\infty} A_n$ is also in Σ, enabling the construction of measurable sets from countably infinite operations. De Morgan's laws then imply closure under countable intersections, as $\bigcap_{n=1}^{\infty} A_n = \left(\bigcup_{n=1}^{\infty} A_n^c\right)^c$, ensuring that the framework accommodates conjunctions of countable collections of events. Finally, the inclusion of the empty set $\emptyset \in \Sigma$ is an axiom that provides a null baseline, ensuring that the σ-algebra is non-empty and can represent the "impossibility" of certain outcomes.

Literature Review: Rao et. al. (2024) [1] investigated approximation theory within Lebesgue measurable function spaces, providing an analysis of operator convergence. They also established a theoretical framework for function approximation in Lebesgue spaces and provided a rigorous study of symmetric properties in function spaces. Mukhopadhyay and Ray (2025) [2] provided a comprehensive introduction to measurable function spaces, with a focus on Lp-spaces and their completeness properties. They also established the fundamental role of Lp-spaces in measure theory and discussed the relationship between continuous function spaces and measurable functions. Szołdra (2024) [3] examined measurable function spaces in quantum mechanics, exploring the role of measurable observables in ergodic theory. They connected functional analysis and measure theory to quantum state evolution and provided a mathematical foundation for quantum machine learning in function spaces. Lee (2025) [10] investigated metric space theory and functional analysis in the context of measurable function spaces in AI models. He formalized how function spaces can model self-referential structures in AI and provided a bridge between measure theory and generative models. Song et. al. (2025) [4] discussed measurable function spaces in the context of urban renewal and performance evaluation. They developed a rigorous evaluation metric using measurable function spaces and explored function space properties in applied data science and urban analytics. Mennaoui et. al. (2025) [5] explored measurable function spaces in the theory of evolution equations, a key concept in functional analysis. They established a rigorous study of measurable operator functions and provided new insights into function spaces and their role in solving differential equations. Pedroza (2024) [6] investigated domain stability in machine learning models using function spaces. He established a formal mathematical relationship between function smoothness and domain adaptation and uses topological and measurable function spaces to analyze stability conditions in learning models. Cerreia-Vioglio and Ok (2024) [7] developed a new integration theory for measurable set-valued functions. They introduced a generalization of integration over Banach-valued functions and established weak compactness properties in measurable function spaces. Averin (2024) [8] applied measurable function spaces to gravitational entropy theory. He provided a rigorous proof of entropy bounds using function space formalism and connected measure theory with relativistic field equations. Potter (2025) [9] investigated measurable function spaces in the context of Fourier analysis and crystallographic structures. He established new results on Fourier transforms of measurable functions and introduced a novel framework for studying function spaces in invariant shift operators.

Measurable spaces are not merely abstract structures but are the backbone of measure theory, probability, and integration. For example, the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ on the real numbers $\mathbb{R}$ is the smallest σ-algebra containing all open intervals (a, b) for $a, b \in \mathbb{R}$. This σ-algebra is pivotal in defining Lebesgue measure, where measurable sets generalize the classical notion of intervals to include sets that are neither open nor closed. Moreover, the construction of a σ-algebra generated by a collection of subsets $\mathcal{C} \subseteq 2^{\mathcal{X}}$, denoted $\sigma(\mathcal{C})$, provides a minimal framework that includes $\mathcal{C}$ and satisfies all σ-algebra properties, enabling the systematic extension of measurability to more complex scenarios. For instance, starting with intervals in $\mathbb{R}$, one can build the Borel σ-algebra, a critical tool in modern analysis.

The structure of a measurable space allows the definition of a measure μ, a function $\mu : \Sigma \to [0, \infty]$ that assigns a non-negative value to each set in Σ, adhering to two key axioms: $\mu(\emptyset) = 0$ and countable additivity, which states that for any disjoint collection $\{A_n\}_{n=1}^{\infty} \subseteq \Sigma$, the measure of their union satisfies $\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n)$. This property is indispensable in extending intuitive notions of length, area, and volume to arbitrary measurable sets, paving the way for the Lebesgue integral. A function $f : \mathcal{X} \to \mathbb{R}$ is then termed Σ-measurable if for every Borel set $B \in \mathcal{B}(\mathbb{R})$, the preimage $f^{-1}(B)$ belongs to Σ. This definition ensures that the function is compatible with the σ-algebra, a necessity for defining integrals and expectation in probability theory.

In summary, measurable spaces represent a highly versatile and mathematically rigorous framework, underpinning vast areas of analysis and probability. Their theoretical depth lies in their ability to systematically handle infinite operations while maintaining closure, consistency, and flexibility for defining measures, measurable functions, and integrals. As such, the rigorous study of measurable spaces is indispensable for advancing modern mathematical theory, providing a bridge between abstract set theory and applications in real-world phenomena.

Let $(\mathcal{X}, \Sigma_X, \mu_X)$ and $(\mathcal{Y}, \Sigma_Y, \mu_Y)$ be measurable spaces. The true risk functional is defined as:

$$\mathcal{R}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y)\, dP(x, y), \tag{1.2}$$

where:

- f belongs to a hypothesis space $\mathcal{F} \subseteq L^p(\mathcal{X}, \mu_X)$.
- $P(x, y)$ is a Borel probability measure over $\mathcal{X} \times \mathcal{Y}$, satisfying $\int_{\mathcal{X}\times\mathcal{Y}} 1\, dP = 1$.

1.1.2 Risk as a Functional

Literature Review: Wang et. al. (2025) [11] developed a mathematical risk model based on functional variational calculus and introduced a loss functional regularization framework that minimizes adversarial risk in deep learning models. They also proposed a game-theoretic interpretation of functional risk in security settings. Duim and Mesquita (2025) [12] extended the inverse reinforcement learning (IRL) framework by defining risk as a functional over probability spaces and used Bayesian functional priors to model risk-sensitive behavior. They also introduced an iterative regularized risk functional minimization approach. Khayat et. al. (2025) [13] established functional Sobolev norms to quantify risk in adversarial settings and introduced a functional risk decomposition technique using deep neural architectures. They also provided an in-depth theoretical framework for risk estimation in adversarially perturbed networks. Agrawal (2025) [14] developed a variational framework for risk as a loss functional and used adaptive weighting of loss functions to enhance generalization in deep learning. He also provided rigorous convergence analysis of risk functional minimization. Hailemichael and Ayalew (2025) [15] used control barrier function (CBF) theory to develop risk-aware deep learning models and modeled risk as a functional on dynamical systems, optimizing stability and robustness. They also introduced a risk-minimizing constrained optimization formulation. Nguyen et.al. (2025) [16] developed a functional metric learning approach for risk-sensitive deep models and used convex optimization techniques to derive functional risk bounds. They also established semi-supervised loss functions for risk-regularized learning. Luo et. al. (2025) [17] introduced a geometric interpretation of risk functionals in deep learning models and used integral transform techniques to approximate risk in real-world vision systems. They also developed a functional approach to adversarial robustness.

The functional $\mathcal{R} : \mathcal{F} \to \mathbb{R}_+$ is Fréchet-differentiable if:

$$\forall f, g \in \mathcal{F}, \quad \mathcal{R}(f + \epsilon g) = \mathcal{R}(f) + \epsilon\langle \nabla\mathcal{R}(f), g\rangle + o(\epsilon), \tag{1.3}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $L^2(\mathcal{X})$. In the field of risk management and decision theory, the concept of a **risk functional** is a mathematical formalization that captures how risk is quantified for a given outcome or state. A risk functional, denoted as $\mathcal{R}$, acts as a map that takes elements from a given space X (which represents the possible outcomes or states) and returns a real-valued risk measure. This risk measure, $\mathcal{R}(x)$, expresses the degree of risk or the adverse outcome associated with a particular element $x \in X$. The space X may vary depending on the context—this could be a space of random variables, trajectories, or more complex function spaces. The risk functional maps x to $\mathbb{R}$, i.e., $\mathcal{R} : X \to \mathbb{R}$, where each $\mathcal{R}(x)$ reflects the risk associated with the specific realization x. One of the most foundational forms of risk functionals is based on the **expectation** of a loss function $L(x)$, where $x \in X$ represents a random variable or state, and $L(x)$ quantifies the loss associated with that state. The risk functional can be expressed as an expected loss, written mathematically as:

$$\mathcal{R}(x) = \mathbb{E}[L(x)] = \int_X L(x)p(x)\,dx \tag{1.4}$$

where $p(x)$ is the probability density function of the outcome x, and the integration is taken over the entire space X. In this setup, $L(x)$ can be any function that measures the severity or unfavorable nature of the outcome x. In a financial context, $L(x)$ could represent the loss function for a portfolio, and $p(x)$ would be the probability density function of the portfolio's returns. In many cases, a specific form of $L(x)$ is used, such as $L(x) = (x - \mu)^2$, where μ is the target or expected value. This choice results in the risk functional representing the **variance** of the outcome x, expressed as:

$$\mathcal{R}(x) = \int_X (x - \mu)^2 p(x)\,dx \tag{1.5}$$

This formulation captures the variability or dispersion of outcomes around a mean value, a common risk measure in applications like portfolio optimization or risk management. Additionally, another widely used class of risk functionals arises from **quantile-based risk measures**, such as **Value-at-Risk (VaR)**, which focuses on the extreme tail behavior of the loss distribution. The VaR at a confidence level $\alpha \in [0, 1]$ is defined as the smallest value l such that the probability of $L(x)$ exceeding l is no greater than $1 - \alpha$, i.e.,

$$P(L(x) \leq l) \geq \alpha \tag{1.6}$$

This defines a threshold beyond which the worst outcomes are expected to occur with probability $1 - \alpha$. Value-at-Risk provides a measure of the worst-case loss under normal circumstances, but it does not provide information about the severity of losses exceeding this threshold. To address this limitation, the **Conditional Value-at-Risk (CVaR)** is introduced, which measures the expected loss given that the loss exceeds the VaR threshold. Mathematically, CVaR at the level α is given by:

$$\mathrm{CVaR}_\alpha(x) = \mathbb{E}[L(x) \mid L(x) \geq \mathrm{VaR}_\alpha(x)] \tag{1.7}$$

This conditional expectation provides a more detailed assessment of the potential extreme losses beyond the VaR threshold. The CVaR is a more comprehensive measure, capturing the tail risk and providing valuable information about the magnitude of extreme adverse events. In cases where the space X represents **trajectories** or paths, such as in the context of continuous-time processes or dynamical systems, the risk functional is often formulated in terms of integrals over time. For example, consider $x(t)$ as a trajectory in the function space $\mathcal{C}([0, T], \mathbb{R}^n)$, the space of continuous functions on the interval $[0, T]$. The risk functional in this case might quantify the total deviation of the trajectory from a reference or target trajectory over time. A typical example could be the total squared deviation, written as:

$$\mathcal{R}(x) = \int_0^T \|x(t) - \bar{x}(t)\|^2 dt \tag{1.8}$$

where $\bar{x}(t)$ represents a reference trajectory and $\|\cdot\|$ is a norm, such as the Euclidean norm. This risk functional quantifies the total deviation (or **energy**) of the trajectory from the target path over the entire time interval, and is used in various applications such as control theory and optimal trajectory planning. A common choice for the norm $\|x(t)\|$ might be $\|x(t)\|^2 = \sum_{i=1}^{n} x_i^2(t)$, where $x_i(t)$ are the components of the trajectory $x(t)$ in $\mathbb{R}^n$. In some cases, the space X of possible outcomes may not be a finite-dimensional vector space, but instead a **Banach space** or a **Hilbert space**, particularly when x represents a more complex object such as a function or a trajectory. For example, the space $\mathcal{C}([0,T],\mathbb{R}^n)$ is a Banach space, and the risk functional may involve the evaluation of integrals over this function space. In such settings, the risk functional can take the form:

$$\mathcal{R}(x) = \int_0^T \|x(t)\|_p^p dt \tag{1.9}$$

where $\|\cdot\|_p$ is the p-norm, and $p \geq 1$. For $p = 2$, this risk functional represents the total **energy** of the trajectory, but other norms can be used to emphasize different types of risks. For instance, the L^∞-norm would focus on the maximum deviation of the trajectory from the target path. The concept of **convexity** plays a significant role in the theory of risk functionals. Convexity ensures that the risk associated with a **convex combination** of two states x_1 and x_2 is less than or equal to the weighted average of the risks of the individual states. Mathematically, for $\lambda \in [0,1]$, convexity demands that:

$$\mathcal{R}(\lambda x_1 + (1-\lambda)x_2) \leq \lambda\mathcal{R}(x_1) + (1-\lambda)\mathcal{R}(x_2) \tag{1.10}$$

This property reflects the diversification effect in risk management, where mixing several states or outcomes generally leads to a reduction in overall risk. Convex risk functionals are particularly important in portfolio theory, where they allow for risk minimization through diversification. For example, if $\mathcal{R}(x)$ represents the variance of a portfolio's returns, then the convexity property ensures that combining different assets will result in a portfolio with lower overall risk than the risk of any individual asset. **Monotonicity** is another important property for risk functionals, ensuring that the risk increases as the outcome becomes more adverse. If x_1 is worse than x_2 according to some partial order, we have:

$$\mathcal{R}(x_1) \geq \mathcal{R}(x_2) \tag{1.11}$$

Monotonicity ensures that the risk functional behaves in a way that aligns with intuitive notions of risk: worse outcomes are associated with higher risk. In financial contexts, this is reflected in the fact that **losses** increase the associated risk measure. Finally, in some applications, the risk functional is derived from perturbation analysis to study how small changes in parameters affect the overall risk. Consider $x(\epsilon)$ as a perturbed trajectory, where ϵ is a small parameter, and the Fréchet derivative of the risk functional with respect to ϵ is given by:

$$\left.\frac{d}{d\epsilon}\mathcal{R}(x(\epsilon))\right|_{\epsilon=0} \tag{1.12}$$

This derivative quantifies the sensitivity of the risk to perturbations in the system and is crucial in the analysis of stability and robustness. Such analyses are essential in areas like **stochastic control** and **optimization**, where it is important to understand how small changes in the model's parameters can influence the risk profile.

Thus, the risk functional is a powerful tool for quantifying and managing uncertainty, and its formulation can be adapted to various settings, from random variables and stochastic processes to continuous trajectories and dynamic systems. The risk functional provides a rigorous mathematical framework for assessing and minimizing risk in complex systems, and its flexibility makes it applicable across a wide range of domains.

1.2 Approximation Spaces for Neural Networks

The neural network hypothesis space $\mathcal{F}_\theta$ is parameterized as:

$$\mathcal{F}_\theta = \{f_\theta : \mathcal{X} \to \mathbb{R} \mid f_\theta(x) = \sum_{j=1}^{n} c_j \sigma(a_j \cdot x + b_j),\, \theta = (c, a, b)\}. \tag{1.13}$$

To analyze its capacity, we rely on:

- **VC-dimension theory** for discrete hypotheses.
- **Rademacher complexity** for continuous spaces:

  $$\mathcal{R}_N(\mathcal{F}) = \mathbb{E}_\epsilon \left[\sup_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \epsilon_i f(x_i) \right], \tag{1.14}$$

  where ϵ_i are i.i.d. Rademacher random variables.

1.2.1 VC-dimension theory for discrete hypotheses

The VC-dimension (Vapnik-Chervonenkis dimension) is a fundamental concept in statistical learning theory that quantifies the capacity of a hypothesis class to fit a range of labelings of a set of data points. The VC-dimension is particularly useful in understanding the generalization ability of a classifier. The theory is important in machine learning, especially when assessing overfitting and the risk of model complexity.

Literature Review: There are several articles that explore the VC-dimension theory for discrete hypotheses very rigorously. N. Bousquet and S. Thomassé (2015) [18] explored in their paper the VC-dimension in the context of graph theory, connecting it to structural properties such as the Erdős-Pósa property. Yıldız and Alpaydin (2009) [19] in their article computed the VC-dimension for decision tree hypothesis spaces, considering both discrete and continuous features. Zhang et. al. (2012) [20] introduced a discretized VC-dimension to bridge real-valued and discrete hypothesis spaces, offering new theoretical tools for complexity analysis. Riondato and Zdonik (2011) [21] adapted VC-dimension theory to database systems, analyzing SQL query selectivity using a theoretical lens. Riggle and Sonderegger (2010) [22] investigated the VC-dimension in linguistic models, focusing on grammar hypothesis spaces. Anderson (2023) [23] provided a comprehensive review of VC-dimension in fuzzy systems, particularly in logic frameworks involving discrete structures. Fox et. al. (2021) [24] proved key conjectures for systems with bounded VC-dimension, offering insights into combinatorial implications. Johnson (2021) [25] discusses binary representations and VC-dimensions, with implications for discrete hypothesis modeling. Janzing (2018) [26] in his paper focuses on hypothesis classes with low VC-dimension in causal inference frameworks. Hüllermeier and Tehrani (2012) [27] in their paper explored the theoretical VC-dimension of Choquet integrals, applied to discrete machine learning models. The book titled "Foundations of Machine Learning" [28] by Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar offers a very good foundational discussion on VC-dimension in the context of statistical learning. Another book titled "Learning Theory: An Approximation Theory Viewpoint" by Felipe Cucker and Ding-Xuan Zhou [29] discusses the role of VC-dimension in approximation theory. Yet another book titled "Understanding Machine Learning: From Theory to Algorithms" by Shai Shalev-Shwartz and Shai Ben-David [30] contains detailed chapters on hypothesis spaces and VC-dimension.

For discrete hypotheses, the VC-dimension theory applies to a class of hypotheses that map a set of input points to binary output labels (typically 0 or 1). The VC-dimension for a hypothesis class refers to the largest set of data points that can be shattered by that class, where "shattering" means that the hypothesis class can realize all possible labelings of these points.

We shall now discuss the **Formal Mathematical Framework**. Let X be a finite or infinite set called the **instance space**, which represents the input space. Consider a hypothesis class H, where each hypothesis $h \in H$ is a function $h : X \to \{0, 1\}$. The function h classifies each element of X into one of two classes: 0 or 1. Given a subset $S = \{x_1, x_2, \dots, x_k\} \subseteq X$, we say that H **shatters** S if for every possible labeling $\vec{y} = (y_1, y_2, \dots, y_k) \in \{0, 1\}^k$, there exists some $h \in H$ such that for all $i \in \{1, 2, \dots, k\}$:

$$h(x_i) = y_i \tag{1.15}$$

In other words, a hypothesis class H shatters S if it can produce every possible binary labeling on the set S. The **VC-dimension** $\mathrm{VC}(H)$ is defined as the size of the largest set S that can be shattered by H:

$$\mathrm{VC}(H) = \sup\{k \mid \exists S \subseteq X, |S| = k, S \text{ is shattered by } H\} \tag{1.16}$$

If no set of points can be shattered, then the VC-dimension is 0. Some Properties of the VC-Dimension are

1. **Shattering Implies Non-empty Hypothesis Class**: If a set S is shattered by H, then H is non-empty. This follows directly from the fact that for each labeling $\vec{y} \in \{0, 1\}^k$, there exists some $h \in H$ that produces the corresponding labeling. Therefore, H must contain at least one hypothesis.

2. **Upper Bound on Shattering**: Given a hypothesis class H, if there exists a set $S \subseteq X$ of size k such that H can shatter S, then any set $S' \subseteq X$ of size greater than k cannot be shattered. This gives us the crucial result that:

$$\mathrm{VC}(H) \geq k \quad \text{if} \quad H \text{ can shatter a set of size } k \tag{1.17}$$

3. **Implication for Generalization** A central result in the theory of **statistical learning** is the connection between VC-dimension and the **generalization error**. Specifically, the **VC-dimension** bounds the ability of a hypothesis class to generalize to unseen data. The higher the VC-dimension, the more complex the hypothesis class, and the more likely it is to **overfit** the training data, leading to poor generalization.

We shall now discuss the VC-Dimension and Generalization Bounds (VC Theorem). The **VC-dimension theorem** (often referred to as **Hoeffding's bound** or the **generalization bound**) provides a probabilistic guarantee on the relationship between the training error and the true error. Specifically, it gives an upper bound on the probability that the generalization error exceeds the empirical error (training error) by more than ϵ.

The **Vapnik-Chervonenkis (VC) dimension** is a fundamental measure of the capacity of a hypothesis class $\mathcal{H}$, and it plays a crucial role in understanding the generalization ability of machine learning models, including neural networks. The VC-dimension of a hypothesis class is defined as the largest number m such that there exists a set of m points that can be **shattered** by $\mathcal{H}$. Formally, a set $S = \{x_1, x_2, \dots, x_m\}$ is shattered by $\mathcal{H}$ if for every possible binary labeling $\{y_1, y_2, \dots, y_m\} \in \{0, 1\}^m$, there exists a hypothesis $h \in \mathcal{H}$ such that $h(x_i) = y_i$ for all i. The VC-dimension, denoted as $VC(\mathcal{H})$, is the supremum of all such m that can be shattered by $\mathcal{H}$. For a neural network with W total parameters (weights and biases), the capacity of the hypothesis class induced by the network architecture depends on the activation function. If the activation function is piecewise linear (such as ReLU), then an upper bound for the VC-dimension is given by

$$VC(\mathcal{H}) = O(W \log W). \tag{1.18}$$

This result is derived from combinatorial arguments involving the number of hyperplane arrangements that a neural network can realize in a given input space. If the activation function is nonlinear (such as sigmoid or tanh), the capacity can increase but remains constrained by the number of parameters. The intuition behind this bound lies in the observation that each neuron introduces a decision boundary in the input space, and the number of such boundaries grows approximately as $W \log W$ rather than W due to dependencies among neurons. When considering **discrete hypotheses**, the hypothesis space $\mathcal{H}$ is restricted to a finite number of functions. This occurs in scenarios where the weights and biases are **quantized** to a finite set of values, or where the activation functions themselves take on a finite number of distinct outputs. If there are K possible distinct values for each parameter and there are W parameters, then the total number of possible functions that the network can realize is at most

$$|\mathcal{H}| \leq K^W. \tag{1.19}$$

From the fundamental relationship between the VC-dimension and the number of hypotheses in a finite hypothesis space, we obtain the bound

$$VC(\mathcal{H}) \leq \log_2 |\mathcal{H}|. \tag{1.20}$$

Applying this result to a neural network with quantized weights and biases, we obtain

$$VC(\mathcal{H}) \leq W \log_2 K. \tag{1.21}$$

Thus, **discretization significantly reduces the VC-dimension** compared to continuous-valued networks, where the number of hypotheses is effectively infinite due to the continuous range of parameters. For binary networks, where each weight is restricted to two possible values (e.g., ± 1), the bound simplifies further to

$$VC(\mathcal{H}) \leq W. \tag{1.22}$$

Since VC-dimension directly affects the generalization error, its role in neural network theory is formalized through the following **uniform convergence bound**. If h is chosen from $\mathcal{H}$ based on an i.i.d. training set of size N, then with probability at least $1 - \delta$, the true error $\ell(h)$ and the empirical error $\hat{\ell}(h)$ satisfy the bound

$$\sup_{h \in \mathcal{H}} |\ell(h) - \hat{\ell}(h)| \leq \sqrt{\frac{VC(\mathcal{H}) \log \frac{N}{VC(\mathcal{H})} + \log \frac{1}{\delta}}{N}}. \tag{1.23}$$

A crucial consequence of this inequality is that for a hypothesis class with a **larger VC-dimension**, a larger number of training samples is required to achieve the same generalization error. Since discrete neural networks have a **lower** VC-dimension than their continuous counterparts, they often generalize better given the same training set size, provided the hypothesis class is sufficiently expressive for the problem at hand. The **shattering coefficient**, also known as the **growth function**, plays an essential role in understanding the VC-dimension of neural networks. The shattering coefficient $S_{\mathcal{H}}(m)$ is defined as the maximum number of dichotomies that $\mathcal{H}$ can realize on any set of m points. If $S_{\mathcal{H}}(m) = 2^m$, then $\mathcal{H}$ can fully shatter an m-point dataset. For neural networks with continuous parameters, the growth function satisfies

$$S_{\mathcal{H}}(m) \leq O(m^{VC(\mathcal{H})}). \tag{1.24}$$

For discrete neural networks, where $|\mathcal{H}|$ is finite, we obtain the bound

$$S_{\mathcal{H}}(m) \leq |\mathcal{H}|, \tag{1.25}$$

which ensures that the number of realizable functions is significantly smaller than in the continuous case, thereby **reducing overfitting potential**. The interplay between VC-dimension and **structural risk minimization (SRM)** further highlights the importance of capacity control in neural networks. Given a hierarchy of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots$, with increasing VC-dimension, the optimal generalization performance is achieved by choosing the class $\mathcal{H}_k$ that minimizes the trade-off between empirical risk and capacity, as given by

$$R(h) \leq \hat{R}(h) + \mathcal{O}\left(\sqrt{\frac{VC(\mathcal{H}_k)\log N}{N}}\right). \tag{1.26}$$

For discrete networks, the practical implication of this bound is that models with **limited precision weights** or **constrained architectures** tend to generalize better than overparameterized networks with unnecessarily high VC-dimension. From an optimization standpoint, discrete neural networks have **a smaller number of local minima** in the loss landscape compared to their continuous counterparts. This is because the number of unique parameter configurations is finite, leading to a more structured and potentially more **tractable optimization problem**. However, the trade-off is that discrete optimization is often **NP-hard**, requiring specialized techniques such as simulated annealing, evolutionary algorithms, or integer programming methods. Thus, VC-dimension theory provides profound insights into the **expressive power**, **generalization ability**, and **optimization complexity** of neural networks. Discrete neural networks exhibit a **reduced VC-dimension** compared to their continuous counterparts, leading to **potentially better generalization**, provided that the model retains sufficient expressivity for the given problem. The **trade-off between expressivity and generalization** is fundamental to designing efficient neural network architectures that perform well in practice.

Let $\mathcal{D}$ be the distribution from which the training data is drawn, and let $e\hat{r}r(h)$ and $err(h)$ represent the **empirical error** and **true error** of a hypothesis $h \in H$, respectively:

$$e\hat{r}r(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{h(x_i)\neq y_i\}} \tag{1.27}$$

$$err(h) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left(h(x) \neq y\right) \tag{1.28}$$

where $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ are i.i.d. (independent and identically distributed) samples from the distribution $\mathcal{D}$. For a hypothesis class H with **VC-dimension** $d = \mathrm{VC}(H)$, with probability at least $1 - \delta$, the following holds for all $h \in H$:

$$|e\hat{r}r(h) - err(h)| \leq \epsilon \tag{1.29}$$

where ϵ is bounded by:

$$\epsilon \leq \sqrt{\frac{8}{n}\left(d\log\left(\frac{2n}{d}\right) + \log\left(\frac{4}{\delta}\right)\right)} \tag{1.30}$$

This result shows that the generalization error (the difference between the true and empirical error) is small with high probability, provided the sample size n is large enough and the VC-dimension d is not too large. The sample complexity n required to guarantee that the generalization error is within ϵ with high probability $1 - \delta$ is given by:

$$n \geq \frac{C}{\epsilon^2}\left(d\log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right) \tag{1.31}$$

where C is a constant depending on the distribution. This bound emphasizes the importance of VC-dimension in controlling the complexity of the hypothesis class. A larger VC-dimension requires a larger sample size to avoid overfitting and ensure reliable generalization. Some Detailed Examples are:

1. **Example 1: Linear Classifiers in $\mathbb{R}^2$**: Consider the hypothesis class H consisting of linear classifiers in $\mathbb{R}^2$. These classifiers are hyperplanes in two dimensions, defined by:

$$h(x) = \text{sign}(w^T x + b) \tag{1.32}$$

   where $w \in \mathbb{R}^2$ is the weight vector and $b \in \mathbb{R}$ is the bias term. The **VC-dimension** of linear classifiers in $\mathbb{R}^2$ is 3. This can be rigorously shown by noting that for any set of 3 points in $\mathbb{R}^2$, the hypothesis class H can shatter these points. In fact, any possible binary labeling of the 3 points can be achieved by some linear classifier. However, for 4 points in $\mathbb{R}^2$, it is impossible to shatter all possible binary labelings (e.g., the four vertices of a convex quadrilateral), meaning the VC-dimension is 3.

2. **Example 2: Polynomial Classifiers of Degree d**: Consider a polynomial hypothesis class in $\mathbb{R}^n$ of degree d. The hypothesis class H consists of polynomials of the form:

$$h(x) = \sum_{i_1, i_2, \ldots, i_n} \alpha_{i_1, i_2, \ldots, i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \tag{1.33}$$

   where the $\alpha_{i_1, i_2, \ldots, i_n}$ are coefficients and $x = (x_1, x_2, \ldots, x_n)$. The **VC-dimension** of polynomial classifiers of degree d in $\mathbb{R}^n$ grows as $O(n^d)$, implying that the complexity of the hypothesis class increases rapidly with both the degree d and the dimension n of the input space.

Neural networks, depending on their architecture, can have very high VC-dimensions. In particular, the **VC-dimension** of a neural network with L layers, each containing N neurons, is typically $O(N^L)$, indicating that the VC-dimension grows exponentially with both the number of neurons and the number of layers. This result provides insight into the complexity of neural networks and their capacity to overfit data when the training sample size is insufficient.

The **VC-dimension** of a hypothesis class is a powerful tool in statistical learning theory. It quantifies the complexity of the hypothesis class by measuring its capacity to shatter sets of points, and it is directly tied to the model's ability to generalize. The **VC-dimension theorem** provides rigorous bounds on the generalization error and sample complexity, giving us essential insights into the trade-off between model complexity and generalization. The theory extends to more complex hypothesis classes such as linear classifiers, polynomial classifiers, and neural networks, where it serves as a critical tool for controlling overfitting and ensuring reliable performance on unseen data.

1.2.2 Rademacher complexity for continuous spaces

Literature Review: Truong (2022) [31] in his article explored how Rademacher complexity impacts generalization error in deep learning, particularly with IID and Markov datasets. Gnecco and Sanguineti (2008) [32] developed approximation error bounds in Reproducing Kernel Hilbert Spaces (RKHS) and functional approximation settings. Astashkin (2010) [33] discusses applications of Rademacher functions in symmetric function spaces and their mathematical structure. Ying and Campbell (2010) [34] applies Rademacher complexity to kernel-based learning problems and support vector machines. Zhu et.al. (2009) [35] examined Rademacher complexity in cognitive models and neural representation learning. Astashkin et al. (2020) [36] investigated how the Rademacher system behaves in function spaces and its role in functional analysis. Sachs et.al. (2023) [37] introduced a refined approach to Rademacher complexity tailored to specific machine learning algorithms. Ma and Wang (2020) [38] investigated Rademacher complexity bounds in deep residual networks. Bartlett and Mendelson (2002) [39] wrote a foundational paper on complexity measures, providing fundamental theoretical insights into generalization bounds. Dzahini and Wild (2024) [40] in their paper extended Rademacher-based complexity to stochastic optimization methods. McDonald and Shalizi (2011) [41] showed using sequential Rademacher complexities for I.I.D process how to control the generalization error of time series models wherein past values of the outcome are used to predict future values.

Let $(\mathcal{X}, \Sigma, \mathcal{D})$ represent a probability space where $\mathcal{X}$ is a measurable space, Σ is a sigma-algebra, and $\mathcal{D}$ is a probability measure. The function class $\mathcal{F} \subset L^\infty(\mathcal{X}, \mathbb{R})$ satisfies:

$$\sup_{f \in \mathcal{F}} \|f\|_\infty < \infty, \tag{1.34}$$

where $\|f\|_\infty = \operatorname{ess\,sup}_{x \in \mathcal{X}} |f(x)|$ denotes the essential supremum. For rigor, $\mathcal{F}$ is assumed measurable in the sense that for every $\epsilon > 0$, there exists a countable subset $\mathcal{F}_\epsilon \subseteq \mathcal{F}$ such that:

$$\sup_{f \in \mathcal{F}} \inf_{g \in \mathcal{F}_\epsilon} \|f - g\|_\infty \leq \epsilon. \tag{1.35}$$

Given $S = \{x_1, x_2, \ldots, x_n\} \sim \mathcal{D}^n$, the empirical measure $\mathbb{P}_n$ is:

$$\mathbb{P}_n(A) = \frac{1}{n} \sum_{i=1}^n \nVdash_{\{x_i \in A\}}, \quad \forall A \in \Sigma. \tag{1.36}$$

The integral under $\mathbb{P}_n$ for $f \in \mathcal{F}$ approximates the population integral under $\mathcal{D}$:

$$\mathbb{P}_n[f] = \frac{1}{n} \sum_{i=1}^n f(x_i), \quad \mathcal{D}[f] = \int_{\mathcal{X}} f(x) \, d\mathcal{D}(x). \tag{1.37}$$

Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be independent Rademacher random variables:

$$\mathbb{P}(\sigma_i = +1) = \mathbb{P}(\sigma_i = -1) = \frac{1}{2}, \quad i = 1, \ldots, n. \tag{1.38}$$

These variables are defined on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ independent of the sample S. The Duality and Symmetrization of Empirical Rademacher Complexity is also very important. The empirical Rademacher complexity of $\mathcal{F}$ with respect to S is:

$$\hat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \sigma_i f(x_i) \right], \tag{1.39}$$

where $\mathbb{E}_{\boldsymbol{\sigma}}$ denotes expectation over $\boldsymbol{\sigma}$. The supremum can be interpreted as a functional dual norm in $L^\infty(\mathcal{X}, \mathbb{R})$, where $\mathcal{F}$ is the unit ball. Using the symmetrization technique, the Rademacher complexity relates to the deviation of $\mathbb{P}_n[f]$ from $\mathcal{D}[f]$:

$$\mathbb{E}_S \sup_{f \in \mathcal{F}} |\mathbb{P}_n[f] - \mathcal{D}[f]| \leq 2\mathfrak{R}_n(\mathcal{F}), \tag{1.40}$$

where:

$$\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_S \left[\hat{\mathfrak{R}}_S(\mathcal{F}) \right]. \tag{1.41}$$

This is derived by first symmetrizing the sample and then invoking Jensen's inequality and the independence of $\boldsymbol{\sigma}$. There are some Complexity Bounds that use Covering Numbers and Entropy that need to be discussed. In Metric Entropy, we let $\|\cdot\|_\infty$ be the metric on $\mathcal{F}$. The covering number $N(\epsilon, \mathcal{F}, \|\cdot\|_\infty)$ satisfies:

$$N(\epsilon, \mathcal{F}, \|\cdot\|_\infty) = \inf\{m \in \mathbb{N} : \exists \{f_1, \ldots, f_m\} \subseteq \mathcal{F}, \forall f \in \mathcal{F}, \exists i, \|f - f_i\|_\infty \leq \epsilon\}. \tag{1.42}$$

Regarding the Dudley's Entropy Integral, For a bounded function class $\mathcal{F}$ (compact under $\|\cdot\|_\infty$):

$$\mathfrak{R}_n(\mathcal{F}) \leq \inf_{\alpha > 0} \left(4\alpha + \frac{12}{\sqrt{n}} \int_\alpha^\infty \sqrt{\log N(\epsilon, \mathcal{F}, \|\cdot\|_\infty)} \, d\epsilon \right). \tag{1.43}$$

There is also a Relation to Talagrand's Concentration Inequality. Talagrand's inequality provides tail bounds for the supremum of empirical processes:

$$\mathbb{P}\left(\sup_{f\in\mathcal{F}}|\mathbb{P}_n[f]-\mathcal{D}[f]|>\epsilon\right)\leq 2\exp\left(-\frac{n\epsilon^2}{2\|f\|_\infty^2}\right), \tag{1.44}$$

reinforcing the link between $\mathfrak{R}_n(\mathcal{F})$ and generalization. There are some Applications in Continuous Function Classes. One example is the RKHS with Gaussian Kernel. For $\mathcal{F}$ as the unit ball of an RKHS with kernel $k(x, x')$, the covering number satisfies:

$$\log N(\epsilon,\mathcal{F},\|\cdot\|_\infty)\sim O\left(\frac{1}{\epsilon^2}\right), \tag{1.45}$$

yielding:

$$\mathfrak{R}_n(\mathcal{F})\sim O\left(\frac{1}{\sqrt{n}}\right). \tag{1.46}$$

For $\mathcal{F}\subseteq H^s(\mathbb{R}^d)$, the covering number depends on the smoothness s and dimension d:

$$\mathfrak{R}_n(\mathcal{F})\sim O\left(\frac{1}{n^{s/d}}\right). \tag{1.47}$$

Rademacher complexity is deeply embedded in modern empirical process theory. Its intricate relationship with measure-theoretic tools, symmetrization, and concentration inequalities provides a robust theoretical foundation for understanding generalization in high-dimensional spaces.

1.2.3 Sobolev Embeddings

Literature Review: Abderachid and Kenza (2024) [42] in their paper investigated fractional Sobolev spaces defined using Riemann-Liouville derivatives and studies their embedding properties. It establishes new continuous embeddings between these fractional spaces and classical Sobolev spaces, providing applications to PDEs. Giang et.al. (2024) [43] introduced weighted Sobolev spaces and derived new Pólya-Szegö type inequalities. These inequalities play a key role in establishing compact embedding results in function spaces equipped with weight functions. Ruiz and Fragkiadaki (2024) [44] provided a novel approach using Haar functions to revisit fractional Sobolev embedding theorems and demonstrated the algebra properties of fractional Sobolev spaces, which are essential in nonlinear analysis. Bilalov et.al. (2025) [45] analyzed compact Sobolev embeddings in Banach function spaces, extending the classical Poincaré and Friedrichs inequalities to this setting and provided applications to function spaces used in modern PDE theory. Cheng and Shao (2025) [46] developed the weighted Sobolev compact embedding theorem for function spaces with unbounded radial potentials and used this result to prove the existence of ground state solutions for fractional Schrödinger-Poisson equations. Wei and Zhang (2025) [47] established a new embedding theorem tailored to variational problems arising in Schrödinger-Poisson equations and used Hardy-Sobolev embeddings to study the zero-mass case, an important case in quantum mechanics. Zhang and Qi (2025) [48] examined the compactness of Sobolev embeddings in the presence of small perturbations in quasilinear elliptic equations and proved multiple solution existence results using variational methods. Xiao and Yue (2025) [49] established a Sobolev embedding theorem for fractional Laplacian function spaces and applied the embedding results to image processing, particularly edge detection. Pesce and Portaro (2025) [50] studied intrinsic Hölder spaces and their connection to fractional Sobolev embeddings and established new embedding results for function spaces relevant to ultraparabolic operators.

The Sobolev embedding theorem states that:

$$W^{k,p}(\mathcal{X})\hookrightarrow C^m(\mathcal{X}), \tag{1.48}$$

if $k - \frac{d}{p} > m$, ensuring $f_\theta \in C^\infty(\mathcal{X})$ for smooth activations σ. For a function $u \in L^p(\Omega)$, its weak derivative $D^\alpha u$ satisfies:

$$\int_\Omega u(x) D^\alpha \phi(x)\, dx = (-1)^{|\alpha|} \int_\Omega v(x)\phi(x)\, dx \quad \forall \phi \in C_c^\infty(\Omega), \tag{1.49}$$

where $v \in L^p(\Omega)$ is the weak derivative. This definition extends the classical notion of differentiation to functions that may not be pointwise differentiable. The Sobolev norm encapsulates both function values and their derivatives:

$$\|u\|_{W^{k,p}(\Omega)} = \left(\sum_{|\alpha| \le k} \|D^\alpha u\|_{L^p(\Omega)}^p \right)^{1/p}. \tag{1.50}$$

Key properties:

- **Semi-norm Dominance**: The $W^{k,p}$-norm is controlled by the seminorm $|u|_{W^{k,p}}$, ensuring sensitivity to high-order derivatives.
- **Poincaré Inequality**: For Ω bounded, $u - u_\Omega$ satisfies:

$$\|u - u_\Omega\|_{L^p} \le C \|Du\|_{L^p}. \tag{1.51}$$

Sobolev spaces $W^{k,p}(\Omega)$ embed into $L^q(\Omega)$ or $C^m(\overline{\Omega})$, depending on k, p, q, and n. These embeddings govern the smoothness and integrability of u and its derivatives. There are several Advanced Theorems on Sobolev Embeddings. They are as follows:

1. **Sobolev Embedding Theorem:** Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with Lipschitz boundary. Then:
   - If $k > n/p$, $W^{k,p}(\Omega) \hookrightarrow C^{m,\alpha}(\overline{\Omega})$ with $m = \lfloor k - n/p \rfloor$ and $\alpha = k - n/p - m$.
   - If $k = n/p$, $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ for $q < \infty$.
   - If $k < n/p$, $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ where $\frac{1}{q} = \frac{1}{p} - \frac{k}{n}$.
2. **Rellich-Kondrachov Compactness Theorem:** The embedding $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ is compact for $q < \frac{np}{n-kp}$. Compactness follows from:
   (a) **Equicontinuity**: $W^{k,p}$-boundedness ensures uniform control over oscillations.
   (b) **Rellich's Selection Principle**: Strong convergence follows from uniform estimates and tightness.

The Proof of Sobolev Embedding starts with the Scaling Analysis.Define $u_\lambda(x) = u(\lambda x)$. Then:

$$\|u_\lambda\|_{L^p(\Omega)} = \lambda^{-n/p} \|u\|_{L^p(\lambda^{-1}\Omega)}. \tag{1.52}$$

For derivatives:

$$\|D^\alpha u_\lambda\|_{L^p(\Omega)} = \lambda^{|\alpha| - n/p} \|D^\alpha u\|_{L^p(\lambda^{-1}\Omega)}. \tag{1.53}$$

The scaling relation $\lambda^{k-n/p}$ aligns with the Sobolev embedding condition $k > n/p$. Sobolev norms in $\mathbb{R}^n$ are equivalent to decay rates of Fourier coefficients:

$$\|u\|_{W^{k,p}} \sim \left(\int_{\mathbb{R}^n} |\xi|^{2k} |\hat{u}(\xi)|^2 \, d\xi \right)^{1/2}. \tag{1.54}$$

For $k > n/p$, Fourier decay implies uniform bounds, ensuring $u \in C^{m,\alpha}$. Interpolation spaces bridge L^p and $W^{k,p}$, providing finer embeddings. Duality: Sobolev embeddings are equivalent to boundedness of adjoint operators in L^q. For $-\Delta u = f$, $u \in W^{2,p}(\Omega)$ ensures $u \in C^{0,\alpha}(\overline{\Omega})$ if $p > n/2$. Sobolev spaces govern variational problems in geometry, e.g., minimal surfaces and harmonic maps. On Ω with fractal boundaries, trace theorems refine Sobolev embeddings.

1.2.4 Rellich-Kondrachov Compactness Theorem

The **Rellich-Kondrachov Compactness Theorem** is one of the most fundamental and deep results in the theory of Sobolev spaces, particularly in the study of functional analysis and the theory of partial differential equations. The theorem asserts the compactness of certain Sobolev embeddings under appropriate conditions on the domain and the function spaces involved. This result is of immense significance in mathematical analysis because it provides a rigorous justification for the fact that bounded sequences in Sobolev spaces, under certain conditions, have strongly convergent subsequences in lower-order normed spaces. In essence, the theorem states that while weak convergence in Sobolev spaces is relatively straightforward due to the Banach-Alaoglu theorem, strong convergence is not always guaranteed. However, under the assumptions of the Rellich-Kondrachov theorem, strong convergence in $L^q(\Omega)$ can indeed be obtained from boundedness in $W^{1,p}(\Omega)$. The compactness property ensured by this theorem is much stronger than mere boundedness or weak convergence and plays a crucial role in proving the existence of solutions to variational problems by ensuring that minimizing sequences possess convergent subsequences in an appropriate function space. The theorem can also be viewed as a generalization of the classical **Arzelà–Ascoli theorem**, extending compactness results to function spaces that involve derivatives.

Literature Review: Lassoued (2026) [51] examined function spaces on the torus and their lack of compactness, highlighting cases where the classical Rellich-Kondrachov result fails. He extended compact embedding results to function spaces with periodic structures. He also discussed trace theorems and regular function spaces in this new context. Chen et.al. (2024) [52] extended the Rellich-Kondrachov theorem to Hörmander vector fields, a class of differential operators that appear in hypoelliptic PDEs. They established a degenerate compact embedding theorem, generalizing previous results in the field. They also provided applications to geometric inequalities, highlighting the role of compact embeddings in PDE theory. Adams and Fournier (2003) [53] in their book provided a complete proof of the Rellich-Kondrachov theorem, along with a discussion of compact embeddings. They also covered function space theory, embedding theorems, and applications in PDEs. Brezis (2010) [54] wrote a highly recommended resource for understanding Sobolev spaces and their compactness properties. The book included applications to variational methods and weak solutions of PDEs. Evans (2022) [55] in his classic PDE textbook includes a discussion of compact Sobolev embeddings, their implications for weak convergence, and applications in variational methods. Maz'ya (2011) [56] provided a detailed treatment of Sobolev space theory, including compact embedding theorems in various settings.

To rigorously state the theorem, we consider a bounded open domain $\Omega \subset \mathbb{R}^n$ with a **Lipschitz boundary**. For $1 \leq p < n$, the theorem asserts that the embedding

$$W^{1,p}(\Omega) \hookrightarrow L^q(\Omega) \tag{1.55}$$

is **compact** whenever $q \leq \frac{np}{n-p}$. More precisely, this means that if $\{u_k\} \subset W^{1,p}(\Omega)$ is a bounded sequence in the Sobolev norm, i.e., there exists a constant $C > 0$ such that

$$\|u_k\|_{W^{1,p}(\Omega)} = \|u_k\|_{L^p(\Omega)} + \|\nabla u_k\|_{L^p(\Omega)} \leq C, \tag{1.56}$$

then there exists a **subsequence** $\{u_{k_j}\}$ and a function $u \in L^q(\Omega)$ such that

$$u_{k_j} \to u \quad \text{strongly in } L^q(\Omega), \tag{1.57}$$

which means that

$$\|u_{k_j} - u\|_{L^q(\Omega)} \to 0 \quad \text{as } j \to \infty. \tag{1.58}$$

To establish this rigorously, we first recall the fact that **bounded sequences in $W^{1,p}(\Omega)$ are weakly precompact**. Since $W^{1,p}(\Omega)$ is a **reflexive Banach space** for $1 < p < \infty$, the Banach-Alaoglu theorem ensures that any bounded sequence $\{u_k\}$ in $W^{1,p}(\Omega)$ has a subsequence (still denoted by $\{u_k\}$) and a function $u \in W^{1,p}(\Omega)$ such that

$$u_k \rightharpoonup u \quad \text{in } W^{1,p}(\Omega). \tag{1.59}$$

This means that for all test functions $\varphi \in W^{1,p'}(\Omega)$, where p' is the Hölder conjugate of p satisfying $\frac{1}{p} + \frac{1}{p'} = 1$, we have

$$\int_\Omega u_k \varphi \, dx \to \int_\Omega u\varphi \, dx, \quad \int_\Omega \nabla u_k \cdot \nabla \varphi \, dx \to \int_\Omega \nabla u \cdot \nabla \varphi \, dx. \tag{1.60}$$

However, weak convergence alone does not imply compactness. To obtain strong convergence in $L^q(\Omega)$, we need additional arguments. This is accomplished using the **Fréchet-Kolmogorov compactness criterion**, which states that a bounded subset of $L^q(\Omega)$ is compact if and only if it is **tight** and **uniformly equicontinuous**. More formally, compactness follows if

1. The sequence $u_k(x)$ does not oscillate excessively at small scales.

2. The sequence $u_k(x)$ does not escape to infinity in a way that prevents strong convergence.

To quantify this, we invoke the **Sobolev-Poincaré inequality**, which states that for $p < n$, there exists a constant C such that

$$\|u - u_\Omega\|_{L^q(\Omega)} \le C\|\nabla u\|_{L^p(\Omega)}, \quad u_\Omega = \frac{1}{|\Omega|} \int_\Omega u(x) \, dx. \tag{1.61}$$

Applying this inequality to $u_k - u$, we obtain

$$\|u_k - u\|_{L^q(\Omega)} \le C\|\nabla(u_k - u)\|_{L^p(\Omega)}. \tag{1.62}$$

Since ∇u_k is weakly convergent in $L^p(\Omega)$, we have

$$\|\nabla u_k - \nabla u\|_{L^p(\Omega)} \to 0. \tag{1.63}$$

Thus,

$$\|u_k - u\|_{L^q(\Omega)} \to 0, \tag{1.64}$$

which establishes the **strong convergence in** $L^q(\Omega)$, completing the proof. The key insight is that compactness arises because the gradients of u_k provide control over the oscillations of u_k, ensuring that the sequence cannot oscillate indefinitely without converging in norm. The crucial role of Sobolev embeddings is to guarantee that even though $W^{1,p}(\Omega)$ does not embed compactly into itself, it does embed compactly into $L^q(\Omega)$ for $q < \frac{np}{n-p}$. This embedding ensures that weak convergence in $W^{1,p}(\Omega)$ implies strong convergence in $L^q(\Omega)$, proving the theorem.

1.2.5 Fréchet-Kolmogorov compactness criterion

The Fréchet-Kolmogorov compactness criterion provides necessary and sufficient conditions for a set $\mathcal{F} \subset L^p(\mathbb{R}^n)$, where $1 \le p < \infty$, to be relatively compact, meaning that any sequence $\{f_n\} \subset \mathcal{F}$ has a strongly convergent subsequence in $L^p(\mathbb{R}^n)$. The criterion asserts that $\mathcal{F}$ is relatively compact in $L^p(\mathbb{R}^n)$ if and only if the following three conditions hold: boundedness in L^p, uniform integrability (tightness), and equicontinuity in an integral sense. The proof consists of establishing both the necessary and sufficient conditions.

To prove necessity, suppose that $\mathcal{F}$ is relatively compact in L^p. Then every sequence $\{f_n\} \subset \mathcal{F}$ has a strongly convergent subsequence. This implies that the norms of the functions in $\mathcal{F}$ are uniformly bounded, since otherwise there would exist a sequence $\{f_n\}$ such that $\|f_n\|_{L^p} \to \infty$, contradicting compactness. Hence, there exists a constant $M > 0$ such that for all $f \in \mathcal{F}$,

$$\|f\|_{L^p} = \left(\int_{\mathbb{R}^n} |f(x)|^p dx \right)^{\frac{1}{p}} \leq M. \tag{1.65}$$

Next, to establish uniform integrability, assume for contradiction that $\mathcal{F}$ is not tight. Then there exists $\varepsilon > 0$ such that for every compact set $K \subset \mathbb{R}^n$, there exists some function $f_K \in \mathcal{F}$ satisfying

$$\int_{\mathbb{R}^n \setminus K} |f_K(x)|^p dx \geq \varepsilon. \tag{1.66}$$

This contradicts relative compactness, as it implies the existence of a sequence $\{f_n\}$ with mass escaping to infinity, preventing any strong convergence in L^p. Hence, for every $\varepsilon > 0$, there exists a compact $K \subset \mathbb{R}^n$ such that

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n \setminus K} |f(x)|^p dx < \varepsilon. \tag{1.67}$$

To establish the necessity of equicontinuity, suppose for contradiction that it does not hold. Then there exists $\varepsilon > 0$ such that for every $\delta > 0$, there is some function $f_\delta \in \mathcal{F}$ and some shift h with $|h| < \delta$ such that

$$\int_{\mathbb{R}^n} |f_\delta(x+h) - f_\delta(x)|^p dx \geq \varepsilon. \tag{1.68}$$

This contradicts compactness, as it implies the existence of arbitrarily oscillatory sequences preventing strong convergence in L^p. Hence, for every $\varepsilon > 0$, there exists $\delta > 0$ such that for all $|h| < \delta$,

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n} |f(x+h) - f(x)|^p dx < \varepsilon. \tag{1.69}$$

To prove sufficiency, assume that $\mathcal{F}$ satisfies boundedness, uniform integrability, and equicontinuity. Consider a sequence $\{f_n\} \subset \mathcal{F}$. The first condition guarantees that the sequence is uniformly bounded in L^p, ensuring weak compactness by Banach-Alaoglu. The second condition ensures that the functions do not escape to infinity, implying tightness. The third condition ensures that the sequence is equicontinuous, preventing high-frequency oscillations. By the *diagonal argument*, we can extract a subsequence $\{f_{n_k}\}$ that is a Cauchy sequence in L^p, ensuring strong convergence. To rigorously show that a subsequence converges, define the *modulus of continuity functional*,

$$\omega_{\mathcal{F}}(\delta) = \sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n} |f(x+h) - f(x)|^p dx, \tag{1.70}$$

which satisfies $\omega_{\mathcal{F}}(\delta) \to 0$ as $\delta \to 0$ due to equicontinuity. Given $\varepsilon > 0$, choose δ such that $\omega_{\mathcal{F}}(\delta) < \frac{\varepsilon}{3}$, and a compact set K such that

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n \setminus K} |f(x)|^p dx < \frac{\varepsilon}{3}. \tag{1.71}$$

By weak compactness, there exists a subsequence f_{n_k} converging weakly in L^p to some f. Applying *Vitali's theorem*, we obtain strong convergence, proving compactness. Thus, the Fréchet-Kolmogorov criterion is fully established.

1.2.6 Sobolev-Poincaré inequality

Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with a Lipschitz boundary. Consider the Sobolev space $W^{1,p}(\Omega)$ for $1 \leq p < n$, which consists of functions $u \in L^p(\Omega)$ whose weak derivatives ∇u also belong to

$L^p(\Omega)$. The Sobolev-Poincaré inequality asserts the existence of a constant $C = C(\Omega, p) > 0$ such that

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)} \tag{1.72}$$

for all $u \in W^{1,p}(\Omega)$, where $p^* = \frac{np}{n-p}$ is the Sobolev conjugate exponent and u_Ω is the mean value of u over Ω, given by

$$u_\Omega = \frac{1}{|\Omega|}\int_\Omega u(x)\,dx. \tag{1.73}$$

To prove this inequality, we first use a representation formula for $u(x)$. Given any two points $x, y \in \Omega$, we consider the fundamental theorem of calculus applied along the segment connecting x to y. Define the parametrization

$$\gamma(t) = x + t(y - x), \quad t \in [0, 1]. \tag{1.74}$$

Applying the fundamental theorem of calculus along this line segment,

$$u(y) - u(x) = \int_0^1 \frac{d}{dt}u(\gamma(t))\,dt. \tag{1.75}$$

By the chain rule,

$$\frac{d}{dt}u(\gamma(t)) = \nabla u(\gamma(t)) \cdot (y - x). \tag{1.76}$$

Substituting this into the integral expression,

$$u(y) - u(x) = \int_0^1 \nabla u(\gamma(t)) \cdot (y - x)\,dt. \tag{1.77}$$

Taking the absolute value and applying Hölder's inequality with conjugate exponents p and $p' = \frac{p}{p-1}$,

$$|u(y) - u(x)| \leq |y - x|\int_0^1 |\nabla u(\gamma(t))|\,dt. \tag{1.78}$$

Now, integrating both sides over $y \in \Omega$, using Fubini's theorem and Minkowski's integral inequality,

$$\int_\Omega |u(y) - u(x)|^{p^*}\,dy \leq \int_\Omega \left(|y - x|\int_0^1 |\nabla u(\gamma(t))|\,dt\right)^{p^*}\,dy. \tag{1.79}$$

Using the properties of integral norms and applying Hölder's inequality again,

$$\left(\int_\Omega |u(y) - u(x)|^{p^*}\,dy\right)^{\frac{1}{p^*}} \leq C\|\nabla u\|_{L^p(\Omega)}. \tag{1.80}$$

By averaging over $x \in \Omega$ and defining the mean-value term appropriately, we obtain

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)}, \tag{1.81}$$

where C depends on Ω and p, completing the proof. Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with a Lipschitz boundary. Consider the Sobolev space $W^{1,p}(\Omega)$ for $1 \leq p < n$, which consists of functions $u \in L^p(\Omega)$ whose weak derivatives ∇u also belong to $L^p(\Omega)$. The Sobolev-Poincaré inequality asserts the existence of a constant $C = C(\Omega, p) > 0$ such that

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)} \tag{1.82}$$

for all $u \in W^{1,p}(\Omega)$, where $p^* = \frac{np}{n-p}$ is the Sobolev conjugate exponent and u_Ω is the mean value of u over Ω, given by

$$u_\Omega = \frac{1}{|\Omega|}\int_\Omega u(x)\,dx. \tag{1.83}$$

To prove this inequality, we first use a representation formula for $u(x)$. Given any two points $x, y \in \Omega$, we consider the fundamental theorem of calculus applied along the segment connecting x to y. Define the parametrization

$$\gamma(t) = x + t(y - x), \quad t \in [0, 1]. \tag{1.84}$$

Applying the fundamental theorem of calculus along this line segment,

$$u(y) - u(x) = \int_0^1 \frac{d}{dt} u(\gamma(t))\, dt. \tag{1.85}$$

By the chain rule,

$$\frac{d}{dt} u(\gamma(t)) = \nabla u(\gamma(t)) \cdot (y - x). \tag{1.86}$$

Substituting this into the integral expression,

$$u(y) - u(x) = \int_0^1 \nabla u(\gamma(t)) \cdot (y - x)\, dt. \tag{1.87}$$

Taking the absolute value and applying Hölder's inequality with conjugate exponents p and $p' = \frac{p}{p-1}$,

$$|u(y) - u(x)| \le |y - x| \int_0^1 |\nabla u(\gamma(t))|\, dt. \tag{1.88}$$

Now, integrating both sides over $y \in \Omega$, using Fubini's theorem and Minkowski's integral inequality,

$$\int_\Omega |u(y) - u(x)|^{p^*}\, dy \le \int_\Omega \left(|y - x| \int_0^1 |\nabla u(\gamma(t))|\, dt \right)^{p^*} dy. \tag{1.89}$$

Using the properties of integral norms and applying Hölder's inequality again,

$$\left(\int_\Omega |u(y) - u(x)|^{p^*}\, dy \right)^{\frac{1}{p^*}} \le C \|\nabla u\|_{L^p(\Omega)}. \tag{1.90}$$

By averaging over $x \in \Omega$ and defining the mean-value term appropriately, we obtain

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \le C \|\nabla u\|_{L^p(\Omega)}, \tag{1.91}$$

where C depends on Ω and p, completing the proof. The consequences and applications are:

1. **Regularity of PDE Solutions:** The Sobolev-Poincaré inequality is crucial in proving the existence and regularity of weak solutions to elliptic PDEs.

2. **Compactness and Rellich-Kondrachov Theorem:** It plays a role in proving the compact embedding of $W^{1,p}(\Omega)$ into $L^q(\Omega)$, which is fundamental in functional analysis.

3. **Control of Function Oscillations:** It quantifies how much a function can deviate from its mean, which is used in various areas of mathematical physics and geometry.

One important extension is the sharp form of the Sobolev-Poincaré inequality, which involves explicit best constants in terms of the domain geometry. Specifically, in some cases, the optimal constant is related to eigenvalues of the Laplace operator or geometric properties such as the diameter or inradius of Ω. Another important extension is the fractional Sobolev-Poincaré inequality, which deals with function spaces like $W^{s,p}(\Omega)$ where s is fractional, incorporating nonlocal effects.

In conclusion, the Sobolev-Poincaré inequality is a powerful mathematical result that provides a precise relationship between function averages and their gradients. It is a cornerstone in modern analysis, with deep implications in PDE theory, functional analysis, and geometric measure theory.

2 Universal Approximation Theorem: Refined Proof

The Universal Approximation Theorem (UAT) is a fundamental result in neural network theory, stating that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to any desired degree of accuracy, provided that an appropriate activation function is used. This theorem has significant implications in machine learning, function approximation, and deep learning architectures.

Literature Review: Hornik et. al. (1989) [57] in their seminal paper rigorously proved that multilayer feedforward neural networks with a single hidden layer and a sigmoid activation function can approximate any continuous function on a compact set. It extends prior results and lays the foundation for the modern understanding of UAT. Cybenko (1989) [58] provided one of the first rigorous proofs of the UAT using the sigmoid function as the activation function. They demonstrated that a single hidden layer network can approximate any continuous function arbitrarily well. Barron (1993) [59] extended UAT by quantifying the approximation error and analyzing the rate of convergence. This work is crucial for understanding the practical efficiency of neural networks. Pinkus (1999) [60] provided a comprehensive survey of UAT from the perspective of approximation theory and also discussed conditions for approximation with different activation functions and the theoretical limits of neural networks. Lu et.al. (2017) [61] investigated how the width of neural networks affects their approximation capability, challenging the notion that deeper networks are always better. They also provided insights into trade-offs between depth and width. Hanin and Sellke (2018) [62] extended UAT to ReLU activation functions, showing that deep ReLU networks achieve universal approximation while maintaining minimal width constraints. Garcıa-Cervera et. al. (2024) [63] extended the universal approximation theorem to set-valued functions and its applications to Deep Operator Networks (DeepONets), which are useful in control theory and PDE modeling. Majee et.al. (2024) [64] explored the universal approximation properties of deep neural networks for solving inverse problems using Markov Chain Monte Carlo (MCMC) techniques. Toscano et. al. (2024) [65] introduced Kurkova-Kolmogorov-Arnold Networks (KKANs), an extension of UAT incorporating Kolmogorov's superposition theorem for improved approximation capabilities. Son (2025) [66] established a new framework for operator learning based on the UAT, providing a theoretical foundation for backpropagation-free deep networks.

2.1 Approximation Using Convolution Operators

Let us begin by considering the convolution operator and its role in approximating functions in the context of the Universal Approximation Theorem (UAT). Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous and bounded function. The convolution of f with a kernel function $\phi : \mathbb{R}^n \to \mathbb{R}$, denoted as $f * \phi$, is defined as

$$(f * \phi)(x) = \int_{\mathbb{R}^n} f(y)\phi(x - y)\, dy. \tag{2.1}$$

The kernel $\phi(x)$ is typically chosen to be smooth, compactly supported, and normalized such that

$$\int_{\mathbb{R}^n} \phi(x)\, dx = 1. \tag{2.2}$$

To approximate f locally, we introduce a scaling parameter $\epsilon > 0$ and define the scaled kernel $\phi_\epsilon(x)$ as

$$\phi_\epsilon(x) = \epsilon^{-n} \phi\left(\frac{x}{\epsilon}\right). \tag{2.3}$$

The factor ϵ^{-n} ensures that $\phi_\epsilon(x)$ remains a probability density function, satisfying

$$\int_{\mathbb{R}^n} \phi_\epsilon(x)\, dx = \int_{\mathbb{R}^n} \phi(x)\, dx = 1. \tag{2.4}$$

The convolution of f with the scaled kernel ϕ_ϵ is given by

$$(f * \phi_\epsilon)(x) = \int_{\mathbb{R}^n} f(y)\phi_\epsilon(x-y)\, dy. \tag{2.5}$$

Performing the change of variables $z = \frac{x-y}{\epsilon}$, we have $y = x - \epsilon z$ and $dy = \epsilon^n dz$. Substituting into the integral, we obtain

$$(f * \phi_\epsilon)(x) = \int_{\mathbb{R}^n} f(x - \epsilon z)\phi(z)\, dz. \tag{2.6}$$

This representation shows that $(f * \phi_\epsilon)(x)$ is a smoothed version of $f(x)$, where the smoothing is controlled by the parameter ϵ. As $\epsilon \to 0$, the kernel $\phi_\epsilon(x)$ becomes increasingly concentrated around x, and we recover $f(x)$ in the limit:

$$\lim_{\epsilon \to 0} (f * \phi_\epsilon)(x) = f(x), \tag{2.7}$$

assuming f is continuous. This result can be rigorously proven using properties of the kernel ϕ, such as its smoothness and compact support, and the dominated convergence theorem, which ensures that the integral converges uniformly to $f(x)$. Now, let us consider the role of convolution operators in the approximation of f by neural networks. A single-layer feedforward neural network is expressed as

$$\hat{f}(x) = \sum_{i=1}^{M} c_i \sigma(w_i^T x + b_i), \tag{2.8}$$

where $c_i \in \mathbb{R}$ are coefficients, $w_i \in \mathbb{R}^n$ are weight vectors, $b_i \in \mathbb{R}$ are biases, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. The activation function $\sigma(w_i^T x + b_i)$ can be interpreted as a localized response function, analogous to the kernel $\phi(x - y)$ in convolution. By drawing an analogy between the two, we can write the neural network approximation as

$$\hat{f}(x) \approx \sum_{i=1}^{M} f(x_i)\phi_\epsilon(x - x_i)\Delta x \tag{2.9}$$

where $\phi_\epsilon(x)$ is interpreted as a parameterized kernel defined by w_i, b_i, and σ, and Δx represents a discretization step. The approximation error $\|f - \hat{f}\|_\infty$ can be decomposed into two components:

$$\|f - \hat{f}\|_\infty \le \|f - f * \phi_\epsilon\|_\infty + \|f * \phi_\epsilon - \hat{f}\|_\infty. \tag{2.10}$$

The term $\|f - f * \phi_\epsilon\|_\infty$ represents the error introduced by smoothing f with the kernel ϕ_ϵ, and it can be made arbitrarily small by choosing ϵ sufficiently small, provided f is regular enough (e.g., Lipschitz continuous). The term $\|f * \phi_\epsilon - \hat{f}\|_\infty$ quantifies the error due to discretization, which vanishes as the number of neurons $M \to \infty$. To rigorously analyze the convergence of $\hat{f}(x)$ to $f(x)$, we rely on the density of neural network approximators in function spaces. The Universal

Approximation Theorem states that, for any continuous function f on a compact domain $\Omega \subset \mathbb{R}^n$ and any $\epsilon > 0$, there exists a neural network $\hat{f}$ with finitely many neurons such that

$$\sup_{x\in\Omega} |f(x) - \hat{f}(x)| < \epsilon. \tag{2.11}$$

This result hinges on the ability of the activation function σ to generate a rich set of basis functions. For example, if $\sigma(x) = \max(0, x)$ (ReLU), the network approximates $f(x)$ by piecewise linear functions. If $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid), the network generates smooth approximations that resemble logistic regression.

In this refined proof of the UAT, convolution operators provide a unifying framework for understanding the smoothing, localization, and discretization processes that underlie neural network approximations. The interplay between $\phi_\epsilon(x)$, $f * \phi_\epsilon(x)$, and $\hat{f}(x)$ reveals the profound mathematical structure that connects classical approximation theory with modern machine learning. This connection not only enhances our theoretical understanding of neural networks but also guides the design of architectures and algorithms for practical applications.

2.1.1 Stone-Weierstrass Application

Literature Review: Rudin (1976) [67] introduced the Weierstrass approximation theorem and proves its generalization, the Stone-Weierstrass theorem. He also discussed the algebraic structure of function spaces and how the theorem ensures the uniform approximation of continuous functions by polynomials. He also presented examples and exercises related to compactness, uniform convergence, and Banach algebra structures. Stein and Shakarchi (2005) [68] extended the Stone-Weierstrass theorem into measure theory and functional analysis. He also proved the theorem in the context of Lebesgue integration. He also discussed how it applies to Hilbert spaces and orthogonal polynomials. He also connected the theorem to Fourier analysis and spectral decomposition. Conway (2019) [69] explored the Stone-Weierstrass theorem in the setting of Banach algebras and C-algebras*. He also extended the theorem to non-commutative function algebras and discussed the operator-theoretic implications of the theorem in Hilbert spaces. He also analyzed the theorem's application to spectral theory. Dieudonné (1981) [70] traced the historical development of functional analysis, including the origins of the Stone-Weierstrass theorem and discussed contributions by Karl Weierstrass and Marshall Stone. He also explored how the theorem influenced topological vector spaces and operator theory and also included perspectives on the axiomatic development of function approximation. Folland (1999) [71] discussed the Stone-Weierstrass theorem in depth with applications to probability theory and ergodic theory and used the theorem to establish the density of algebraic functions in measure spaces He also connected the Stone-Weierstrass theorem to functional approximation in Lp spaces. He also explored the interplay between the Stone-Weierstrass theorem and the Hahn-Banach theorem. Sugiura (2024) [72] extended the Stone-Weierstrass theorem to the study of reservoir computing in machine learning and proved that certain neural networks can approximate functions uniformly under the assumptions of the theorem. He bridges classical functional approximation with modern AI and deep learning. Liu et al. (2024) [73] investigated the Stone-Weierstrass theorem in normed module settings and used category theory to generalize function approximation results. He also extended the theorem beyond real-valued functions to structured mathematical objects. Martinez-Barreto (2025) [74] provided a modern formulation of the theorem with rigorous proof and reviewed applications in operator algebras and topology. He also discussed open problems related to function approximation. Chang and Wei (2024) [75] used the Stone-Weierstrass theorem to derive new operator inequalities and applied the theorem to functional analysis in quantum mechanics. Caballer et al. (2024) [76] investigated cases where the Stone-Weierstrass theorem fails and provided counterexamples and refined conditions for uniform approximation. Chen (2024) [77] extended the Stone-Weierstrass theorem to generalized function spaces and introduced a new class of uniform topological algebras. Rafiei and Akbarzadeh-T (2024)

[78] used the Stone-Weierstrass theorem to analyze function approximation in fuzzy logic systems and explored the applications in control systems and AI.

The **Stone-Weierstrass Theorem** serves as a cornerstone in functional analysis, bridging the algebraic structure of continuous functions with approximation theory. This theorem, when applied to the **Universal Approximation Theorem (UAT)**, provides a rigorous foundation for asserting that neural networks can approximate any continuous function defined on a compact set. To understand this connection in its most scientifically and mathematically rigorous form, we must carefully analyze the algebra of continuous functions on a compact Hausdorff space and the role of neural networks in approximating these functions, ensuring that all mathematical nuances are explored with extreme precision. Let X be a compact Hausdorff space, and let $C(X)$ represent the space of continuous real-valued functions on X. The **supremum norm** $\|f\|_\infty$ for a function $f \in C(X)$ is defined as:

$$\|f\|_\infty = \sup_{x \in X} |f(x)| \tag{2.12}$$

This supremum norm is critical in defining the proximity between continuous functions, as we seek to approximate any function $f \in C(X)$ by a function g from a subalgebra $A \subset C(X)$. The **Stone-Weierstrass theorem** guarantees that if the subalgebra A satisfies two essential properties—(1) it contains the constant functions, and (2) it separates points—then the closure of A in the supremum norm will be the entire space $C(X)$. To formalize this, we define the **point separation property** as follows: for every pair of distinct points $x_1, x_2 \in X$, there exists a function $h \in A$ such that $h(x_1) \neq h(x_2)$. This condition ensures that functions from A are sufficiently "rich" to distinguish between different points in X. Mathematically, this is expressed as:

$$\exists h \in A \text{ such that } h(x_1) \neq h(x_2) \quad \forall x_1, x_2 \in X, x_1 \neq x_2 \tag{2.13}$$

Given these two properties, the Stone-Weierstrass theorem asserts that for any continuous function $f \in C(X)$ and any $\epsilon > 0$, there exists an element $g \in A$ such that:

$$\|f - g\|_\infty < \epsilon \tag{2.14}$$

This result ensures that any continuous function on a compact Hausdorff space can be approximated arbitrarily closely by functions from a sufficiently rich subalgebra. In the context of the **Universal Approximation Theorem (UAT)**, we seek to apply the Stone-Weierstrass theorem to the approximation capabilities of neural networks. Let $K \subseteq \mathbb{R}^n$ be a compact subset, and let $f \in C(K)$ be a continuous function defined on this set. A feedforward neural network with a non-linear activation function σ has the form:

$$\hat{f}_\theta(x) = \sum_{i=1}^{N} w_i \sigma(\langle \mathbf{w}_i, x \rangle + b_i) \tag{2.15}$$

where $\langle \mathbf{w}_i, x \rangle$ represents the inner product between the weight vector $\mathbf{w}_i$ and the input x, and b_i represents the bias term. The activation function σ is typically non-linear (such as the sigmoid or ReLU function), and the parameters $\theta = \{w_i, b_i\}_{i=1}^N$ are the weights and biases of the network. The function $\hat{f}_\theta(x)$ is a weighted sum of the non-linear activations applied to the affine transformations of x.

We now explore the connection between neural networks and the Stone-Weierstrass theorem. A critical observation is that the set of functions defined by a neural network with non-linear activation is a subalgebra of $C(K)$ provided the activation function σ is sufficiently rich in its non-linearity. This non-linearity ensures that the network can separate points in K, meaning that for any two distinct points $x_1, x_2 \in K$, there exists a network function $\hat{f}_\theta$ that takes distinct values at these points. This satisfies the point separation condition required by the Stone-Weierstrass theorem.

To formalize this, consider two distinct points $x_1, x_2 \in K$. Since σ is non-linear, the function $\hat{f}_\theta(x)$ with appropriately chosen weights and biases will satisfy:

$$\hat{f}_\theta(x_1) \neq \hat{f}_\theta(x_2) \tag{2.16}$$

Thus, the algebra of neural network functions satisfies the point separation property. By applying the Stone-Weierstrass theorem, we conclude that this algebra is dense in $C(K)$, meaning that for any continuous function $f \in C(K)$ and any $\epsilon > 0$, there exists a neural network function $\hat{f}_\theta$ such that:

$$\|f(x) - \hat{f}_\theta(x)\|_\infty < \epsilon \quad \forall x \in K \tag{2.17}$$

This rigorous result shows that neural networks with a non-linear activation function can approximate any continuous function on a compact set arbitrarily closely in the supremum norm, thereby proving the Universal Approximation Theorem. To further explore this, consider the error term:

$$\|f(x) - \hat{f}_\theta(x)\|_\infty \tag{2.18}$$

For a given function f and a compact set K, this error term can be made arbitrarily small by increasing the number of neurons in the hidden layer of the neural network. This increases the capacity of the network, effectively enlarging the subalgebra of functions generated by the network, thereby improving the approximation. As the number of neurons increases, the network's ability to approximate any function from $C(K)$ becomes increasingly precise, which aligns with the conclusion of the Stone-Weierstrass theorem that the network functions form a dense subalgebra in $C(K)$. Thus, the Universal Approximation Theorem, derived through the Stone-Weierstrass theorem, rigorously proves that neural networks can approximate any continuous function on a compact set to any desired degree of accuracy. The combination of the non-linearity of the activation function and the architecture of the neural network guarantees that the network can generate a dense subalgebra of continuous functions, ultimately allowing it to approximate any function from $C(K)$. This result not only formalizes the approximation power of neural networks but also provides a deep theoretical foundation for understanding their capabilities as universal approximators.

2.2 Depth vs. Width: Capacity Analysis

2.2.1 Bounding the Expressive Power

The Kolmogorov-Arnold Superposition Theorem is a foundational result in the mathematical analysis of multivariate continuous functions and their decompositions, providing a framework that underpins the expressive power of neural networks. It asserts that any continuous multivariate function can be expressed as a finite composition of continuous univariate functions and addition. It was first conjectured by Andrey Kolmogorov in 1956 and later rigorously proved by Vladimir Arnold in 1957. Formally, the theorem guarantees that any continuous multivariate function $f : [0,1]^n \to \mathbb{R}$ can be represented as a finite composition of continuous univariate functions Φ_q and ψ_{pq}. Specifically, for $f(x_1, x_2, \ldots, x_n)$, there exist functions $\Phi_q : \mathbb{R} \to \mathbb{R}$ and $\psi_{pq} : \mathbb{R} \to \mathbb{R}$, such that

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{pq}(x_p) \right), \tag{2.19}$$

where the functions $\psi_{pq}(x_p)$ encode the univariate projections of the input variables x_p, and the outer functions Φ_q aggregate these projections into the final output. This decomposition highlights a fundamental property of multivariate continuous functions: their expressiveness can be captured through hierarchical compositions of simpler, univariate components.

Literature Review: There are some Classical References on the Kolmogorov-Arnold Superposition Theorem (KST). Kolmogorov (1957) [79] in his Foundational Paper on KST established that any continuous function of several variables can be represented as a superposition of continuous functions of a single variable and addition. This was groundbreaking because it provided a universal function decomposition method, independent of inner-product spaces. He proved that there exist functions ϕ_q and ψ_q such that any function $f(x_1, x_2, \ldots, x_n)$ can be expressed as:

$$f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \phi_q \left(\sum_{p=1}^{n} \psi_{qp}(x_p) \right) \tag{2.20}$$

where the ψ_{qp} are univariate functions. Kolmogorov provided a mathematical basis for approximation theory and neural networks, influencing modern machine learning architectures. Arnold (1963) [80] refined Kolmogorov's theorem by proving that one can restrict the superposition to functions of at most two variables instead of one. Arnold's formulation led to the **Kolmogorov-Arnold representation**:

$$f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \phi_q \left(x_q + \sum_{p=1}^{n} \psi_{qp}(x_p) \right) \tag{2.21}$$

making the theorem more suitable for practical computations. Arnold strengthened the expressivity of neural networks, inspiring alternative function representations in high-dimensional settings. Lorentz (2008) [81] in his book discusses the significance of KST in approximation theory and constructive mathematics. He provided error estimates for approximating multivariate functions using Kolmogorov-type decompositions. He showed how KST fits within Bernstein approximation theory. He helped frame KST in the context of function approximation, bridging it to computational applications. Building on this theoretical foundation, Hornik et. al. (1989) [57] demonstrated that multilayer feedforward networks are universal approximators, meaning that neural networks with a single hidden layer can approximate any continuous function. This work bridged the gap between the Kolmogorov-Arnold theorem and practical neural network design, providing a rigorous justification for the use of deep architectures. Pinkus (1999) [60] analyzed the role of **KST in multilayer perceptrons (MLPs)**, showing how it influences function expressibility in neural networks. He demonstrated that feedforward neural networks can approximate arbitrary functions using Kolmogorov superposition. He also provided bounds on network depth and width required for universal approximation. He played a crucial role in understanding the theoretical power of deep learning. In more recent years, Montúfar, Pascanu, Cho, and Bengio (2014) [440] explored the expressive power of deep neural networks by analyzing the number of linear regions they can represent. Their work provided a modern perspective on the Kolmogorov-Arnold theorem, showing how depth enhances the ability of networks to model complex functions. Schmidt-Hieber (2020) [441] rigorously analyzed the approximation properties of deep ReLU networks, demonstrating their efficiency in approximating high-dimensional functions and further connecting the Kolmogorov-Arnold theorem to modern deep learning practices. Yarotsky (2017) [442] complemented this by providing explicit error bounds for approximating functions using deep ReLU networks, offering insights into how depth and activation functions influence approximation accuracy. Telgarsky (2016) [443] contributed to this body of work by rigorously proving that deeper networks can represent functions more efficiently than shallow ones, aligning with the hierarchical decomposition suggested by the Kolmogorov-Arnold theorem. This work provided theoretical insights into why depth is crucial in modern neural networks. Lu et. al. (2017) [444] explored the expressive power of neural networks from the perspective of width rather than depth, showing how width can also play a critical role in function approximation. This complemented the Kolmogorov-Arnold theorem by offering a more nuanced understanding of network design. Finally, Zhang et. al. (2021) [445] provided a rigorous analysis of how deep learning models generalize, which is closely related to their ability to approximate complex functions. While not directly about the Kolmogorov-Arnold theorem, their work contextualized these theoretical insights within the broader framework of generalization in deep learning, offering practical implications for the design and training of neural networks.

There are several very recent contributions in the Kolmogorov-Arnold Superposition Theorem (KST) (2024–2025). Guilhoto and Perdikaris (2024) [82] explored how KST can be reformulated using deep learning architectures. They proposed Kolmogorov-Arnold Networks (KANs), a new type of neural network inspired by KST. They showed that KANs outperform traditional feedforward networks in function approximation tasks. They also provided empirical evidence of KAN efficiency in real-world datasets. They also introduced a new paradigm in machine learning, making function decomposition more interpretable. Alhafiz, M. R. et al. (2025) [83] applied KST-based networks to turbulence modeling in fluid mechanics. They demonstrated how KANs improve predictive accuracy for Navier-Stokes turbulence models. They showed a reduction in computational complexity compared to classical turbulence models. They also developed a data-driven turbulence modeling framework leveraging KST. They advanced machine learning applications in computational fluid dynamics (CFD). Lorencin, I. et al. (2024) [84] used KST-inspired neural networks for predicting propulsion system parameters in ships. They implemented KANs to model hybrid ship propulsion (Combined Diesel-Electric and Gas - CODLAG) and demonstrated a highly accurate prediction model for propulsion efficiency. They also provided a new benchmark dataset for ship propulsion research. They extended KST applications to naval engineering & autonomous systems.

Paper	**Main Contribution**	**Impact**
Kolmogorov (1957)	Original KST theorem	Laid foundation for function decomposition
Arnold (1963)	Refinement using 2-variable functions	Made KST more practical for computation
Lorentz (2008)	KST in approximation theory	Linked KST to function approximation erro
Pinkus (1999)	KST in neural networks	Theoretical basis for deep learning
Perdikaris (2024)	Deep learning reinterpretation	Proposed Kolmogorov-Arnold Networks
Alhafiz (2025)	KST-based turbulence modeling	Improved CFD simulations
Lorencin (2024)	KST in naval propulsion	Optimized ship energy efficiency

In the context of neural networks, this result establishes the theoretical universality of function approximation. A neural network with a single hidden layer approximates a function $f(x_1, x_2, \ldots, x_n)$ by representing it as

$$f(x_1, x_2, \ldots, x_n) \approx \sum_{i=1}^{W} a_i \sigma \left(\sum_{j=1}^{n} w_{ij} x_j + b_i \right), \tag{2.22}$$

where W is the width of the hidden layer, σ is a nonlinear activation function, w_{ij} are weights, b_i are biases, and a_i are output weights. The expressive power of such shallow networks depends critically on the width W, as the universal approximation theorem ensures that $W \to \infty$ suffices to approximate any continuous function arbitrarily well. However, for a fixed approximation error $\epsilon > 0$, the required width grows exponentially with the input dimension n, satisfying a lower bound of

$$W \geq C \cdot \epsilon^{-n}, \tag{2.23}$$

where C depends on the function's Lipschitz constant. This exponential dependence, sometimes called the "curse of dimensionality," underscores the inefficiency of shallow architectures in capturing high-dimensional dependencies.

The advantage of depth becomes apparent when we consider deep neural networks, which utilize hierarchical representations. A deep network with D layers and width W per layer constructs a function as a composition of layer-wise transformations:

$$h^{(k)} = \sigma \left(W^{(k)} h^{(k-1)} + b^{(k)} \right), \quad h^{(0)} = x, \tag{2.24}$$

where $h^{(k)}$ denotes the output of the k-th layer, $W^{(k)}$ is the weight matrix, $b^{(k)}$ is the bias vector, and σ is the nonlinear activation. The final output of the network is then given by

$$f(x) \approx h^{(D)} = \sigma\left(W^{(D)} h^{(D-1)} + b^{(D)}\right). \tag{2.25}$$

The depth D of the network allows it to approximate hierarchical compositions of functions. For example, if a target function $f(x)$ has a compositional structure

$$f(x) = g_1 \circ g_2 \circ \cdots \circ g_D(x), \tag{2.26}$$

where each g_i is a simple function, the depth D directly corresponds to the number of nested transformations. This compositional hierarchy enables deep networks to approximate functions efficiently, achieving a reduction in the required parameter count. The approximation error ϵ for a deep network decreases polynomially with D, satisfying

$$\epsilon \leq O\left(\frac{1}{D^2}\right), \tag{2.27}$$

which is exponentially more efficient than the error scaling for shallow networks. In light of the Kolmogorov-Arnold theorem, the decomposition

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{pq}(x_p) \right) \tag{2.28}$$

demonstrates how deep networks align naturally with the structure of multivariate functions. The inner functions ψ_{pq} capture local dependencies, while the outer functions Φ_q aggregate these into a global representation. This layered decomposition mirrors the depth-based structure of neural networks, where each layer learns a specific aspect of the function's complexity. Finally, the parameter count in a deep network with D layers and width W per layer is given by

$$P \leq O(D \cdot W^2), \tag{2.29}$$

whereas a shallow network requires

$$P \geq O(W^n) \tag{2.30}$$

parameters for the same approximation accuracy. This exponential difference in parameter count illustrates the superior efficiency of deep architectures, particularly for high-dimensional functions. By leveraging the hierarchical decomposition inherent in the Kolmogorov-Arnold theorem, deep networks achieve expressive power that scales favorably with both dimension and complexity.

2.2.2 Fourier Analysis of Expressivity

Literature Review: Juárez-Osorio et. al. (2024) [215] applied Fourier analysis to design quantum convolutional neural networks (QCNNs) for time series forecasting. The Fourier series decomposition helps analyze and optimize expressivity in quantum architectures, making QCNNs better at capturing periodic and non-periodic structures in data. Umeano and Kyriienko (2024) [216] introduced Fourier-based quantum feature maps that transform classical data into quantum states with enhanced expressivity. The Fourier transform plays a central role in mapping high-dimensional data efficiently while maintaining interpretability. Liu et. al. (2024) [217] extended Graph Convolutional Networks (GCNs) by integrating Fourier analysis and spectral wavelets to improve graph expressivity. It bridges the gap between frequency-domain analysis and graph embeddings, making GCNs more effective for complex data structures. Vlasic (2024) [218] presented a Fourier series-inspired feature mapping technique to encode classical data into quantum circuits. It demonstrates how Fourier coefficients can enhance the representational capacity of quantum models, leading to better compression and generalization. Kim et. al. (2024) [219] introduced Neural Fourier Modelling (NFM), a novel approach to representing time-series data compactly while preserving its expressivity. It outperforms traditional models like Short-Time Fourier Transform (STFT) in retaining long-term dependencies. Xie et. al. (2024) [220] explored how Fourier basis functions can be used to enhance the expressivity of tensor networks while maintaining computational efficiency. It establishes trade-offs between expressivity and model complexity in machine learning architectures. Liu et. al. (2024) [221] integrated spectral modulation and Fourier transforms into implicit neural representations for text-to-image synthesis. Fourier analysis improves global coherence while preserving local expressivity in generative models. Zhang (2024) [222] demonstrated how Fourier and Lock-in spectrum techniques can represent long-term variations in mechanical signals. The Fourier-based decomposition allows for more expressive representations of mechanical failures and degradation. Hamed and Lachiri (2024) [223] applied Fourier transformations to speech synthesis models, improving their ability to transfer expressive content from text to speech. Fourier series allows capturing prosody, rhythm, and tone variations effectively. Lehmann et. al. (2024) [224] integrated Fourier-based deep learning models for seismic activity prediction. It explores the expressivity of Fourier Neural Operators (FNOs) in capturing wave propagations in different geological environments.

The Fourier analysis of expressivity in neural networks seeks to rigorously quantify how neural architectures, characterized by their depth and width, can approximate functions through the decomposition of those functions into their Fourier spectra. Consider a square-integrable function $f : \mathbb{R}^d \to \mathbb{R}$, for which the Fourier transform is defined as

$$\hat{f}(\boldsymbol{\xi}) = \int_{\mathbb{R}^d} f(\mathbf{x}) e^{-i2\pi\boldsymbol{\xi}\cdot\mathbf{x}}\, d\mathbf{x} \tag{2.31}$$

where $\boldsymbol{\xi} \in \mathbb{R}^d$ represents the frequency. The inverse Fourier transform reconstructs the function as

$$f(\mathbf{x}) = \int_{\mathbb{R}^d} \hat{f}(\boldsymbol{\xi}) e^{i2\pi\boldsymbol{\xi}\cdot\mathbf{x}}\, d\boldsymbol{\xi} \tag{2.32}$$

The magnitude $|\hat{f}(\boldsymbol{\xi})|$ reflects the energy contribution of the frequency $\boldsymbol{\xi}$ to f. Neural networks approximate f by capturing its Fourier spectrum, but the architecture fundamentally governs how efficiently this approximation can be achieved, especially in the presence of high-frequency components. For shallow networks with one hidden layer and a finite number of neurons, the universal approximation theorem establishes that

$$f(\mathbf{x}) \approx \sum_{i=1}^{n} a_i \phi(\mathbf{w}_i \cdot \mathbf{x} + b_i) \tag{2.33}$$

where ϕ is the activation function, $\mathbf{w}_i \in \mathbb{R}^d$ are weights, $b_i \in \mathbb{R}$ are biases, and $a_i \in \mathbb{R}$ are coefficients. The Fourier transform of this representation can be expressed as

$$\hat{f}(\boldsymbol{\xi}) \approx \sum_{i=1}^{n} a_i \hat{\phi}(\boldsymbol{\xi}) e^{-i2\pi\boldsymbol{\xi}\cdot\mathbf{b}_i} \tag{2.34}$$

where $\hat{\phi}(\boldsymbol{\xi})$ denotes the Fourier transform of the activation function. For smooth activation functions like sigmoid or tanh, $\hat{\phi}(\boldsymbol{\xi})$ decays exponentially as $\|\boldsymbol{\xi}\| \to \infty$, limiting the network's ability to approximate functions with high-frequency content unless the width n is exceedingly large. Specifically, the Fourier coefficients decay as

$$|\hat{f}(\boldsymbol{\xi})| \sim e^{-\beta\|\boldsymbol{\xi}\|} \tag{2.35}$$

where $\beta > 0$ depends on the smoothness of ϕ. This restriction implies that shallow networks are biased toward low-frequency functions unless their width scales exponentially with the input dimension d. Deep networks, on the other hand, leverage their hierarchical structure to overcome these limitations. A deep network with L layers recursively composes functions, producing an output of the form

$$f(\mathbf{x}) = \phi_L(\mathbf{W}^{(L)}\phi_{L-1}(\mathbf{W}^{(L-1)} \cdots \phi_1(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \cdots) + \mathbf{b}^{(L)}) \tag{2.36}$$

where ϕ_l is the activation function at layer l, $\mathbf{W}^{(l)}$ are weight matrices, and $\mathbf{b}^{(l)}$ are bias vectors. The Fourier transform of this composition can be analyzed iteratively. If $\mathbf{h}^{(l)} = \phi_l(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$ represents the output of the l-th layer, then

$$\widehat{\mathbf{h}^{(l)}}(\boldsymbol{\xi}) = \hat{\phi}_l(\boldsymbol{\xi}) * \widehat{\mathbf{W}^{(l)}\mathbf{h}^{(l-1)}}(\boldsymbol{\xi}) \tag{2.37}$$

where $*$ denotes convolution and $\hat{\phi}_l$ is the Fourier transform of the activation function. The recursive application of this convolution amplifies high-frequency components, enabling deep networks to approximate functions whose Fourier spectra exhibit polynomial decay. Specifically, the Fourier coefficients of a deep network decay as

$$|\hat{f}(\boldsymbol{\xi})| \sim \|\boldsymbol{\xi}\|^{-\alpha L} \tag{2.38}$$

where α depends on the activation function. This is in stark contrast to the exponential decay observed in shallow networks.

The activation function plays a pivotal role in shaping the Fourier spectrum of neural networks. For example, the rectified linear unit (ReLU) $\phi(x) = \max(0, x)$ introduces significant high-frequency components into the network. The Fourier transform of the ReLU activation is given by

$$\hat{\phi}(\xi) = \frac{1}{2\pi i \xi} \tag{2.39}$$

which decays more slowly than the Fourier transforms of smooth activations. Consequently, ReLU-based networks are particularly effective at approximating functions with oscillatory behavior. To illustrate, consider the function

$$f(\mathbf{x}) = \sin(2\pi\boldsymbol{\xi} \cdot \mathbf{x}) \tag{2.40}$$

A shallow network requires an exponentially large number of neurons to approximate f when $\|\boldsymbol{\xi}\|$ is large, but a deep network can achieve the same approximation with polynomially fewer parameters by leveraging its hierarchical structure. The expressivity of deep networks can be further quantified by considering their ability to approximate bandlimited functions, i.e., functions f whose Fourier spectra are supported on $\|\boldsymbol{\xi}\| \leq \omega_{\max}$. For a shallow network with width n, the required number of neurons scales as

$$n \sim (\omega_{\max})^d \tag{2.41}$$

where d is the input dimension. In contrast, for a deep network with depth L, the width scales as

$$n \sim (\omega_{\max})^{d/L} \tag{2.42}$$

reflecting the exponential efficiency of depth in distributing the approximation of frequency components across layers. For example, if $f(\mathbf{x}) = \cos(2\pi\boldsymbol{\xi} \cdot \mathbf{x})$ with $\|\boldsymbol{\xi}\| = \omega_{\max}$, a deep network requires significantly fewer parameters than a shallow network to approximate f to the same accuracy. A neural network with an input $\mathbf{x} \in \mathbb{R}^d$ and output $f(\mathbf{x})$ can be expressed as a sum of nonlinearly transformed weighted combinations of the input. Mathematically, this can be written as

$$f(\mathbf{x}) = \sum_{i=1}^{m} a_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i), \tag{2.43}$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{w}_i \in \mathbb{R}^d$ are the weight vectors, $b_i \in \mathbb{R}$ are the biases, and a_i are the output weights. To understand the spectral properties of $f(\mathbf{x})$, it is necessary to analyze the Fourier transform of the activation function $\sigma(\mathbf{w}_i^T\mathbf{x} + b_i)$, as the network output consists of a superposition of such transformed functions. The Fourier transform of $f(\mathbf{x})$ is given by

$$\hat{f}(\mathbf{k}) = \int_{\mathbb{R}^d} f(\mathbf{x}) e^{-2\pi i \mathbf{k}\cdot\mathbf{x}} d\mathbf{x}. \tag{2.44}$$

Substituting the definition of $f(\mathbf{x})$, we obtain

$$\hat{f}(\mathbf{k}) = \sum_{i=1}^{m} a_i \int_{\mathbb{R}^d} \sigma(\mathbf{w}_i^T\mathbf{x} + b_i) e^{-2\pi i \mathbf{k}\cdot\mathbf{x}} d\mathbf{x}. \tag{2.45}$$

The key term in this expression is the Fourier transform of the activation function $\sigma(\mathbf{w}^T\mathbf{x} + b)$, which we denote as

$$\hat{\sigma}(\mathbf{k}) = \int_{\mathbb{R}^d} \sigma(\mathbf{w}^T\mathbf{x} + b) e^{-2\pi i \mathbf{k}\cdot\mathbf{x}} d\mathbf{x}. \tag{2.46}$$

The decay rate of $\hat{\sigma}(\mathbf{k})$ determines the extent to which different frequency components are retained in the Fourier spectrum of the neural network. If $\sigma(x)$ is a smooth function, then $\hat{\sigma}(\mathbf{k})$ decays rapidly for large $\|\mathbf{k}\|$, implying that high-frequency components are suppressed. If $\sigma(x)$ is piecewise continuous or non-differentiable at some points, the decay rate of $\hat{\sigma}(\mathbf{k})$ is slower, allowing the network to capture higher frequencies. To quantify the decay properties of the Fourier transform of an activation function, consider the case of the ReLU activation function, defined as

$$\sigma(x) = \max(0, x). \tag{2.47}$$

Its Fourier transform is given by

$$\hat{\sigma}(k) = \frac{1}{(2\pi i k)^2}, \tag{2.48}$$

which exhibits a power-law decay. In contrast, the sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.49}$$

has a Fourier transform that decays exponentially,

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.50}$$

This strong decay implies that networks with sigmoid activation functions are biased toward learning only low-frequency components. Similarly, the hyperbolic tangent activation function $\tanh(x)$ also exhibits an exponential spectral decay. For a more general class of activation functions, the decay rate of $\hat{\sigma}(k)$ can be estimated as

$$|\hat{\sigma}(k)| \leq C e^{-\alpha|k|} \tag{2.51}$$

for some constants $C, \alpha > 0$, implying that the function is a strong low-pass filter. If $\sigma(x)$ is piecewise smooth but non-differentiable at certain points (such as ReLU), then the Fourier transform satisfies a power-law decay

$$|\hat{\sigma}(k)| \leq C|k|^{-p} \tag{2.52}$$

for some $p > 0$. A particularly interesting case arises when the activation function is sinusoidal, such as $\sigma(x) = \sin(x)$, for which the Fourier transform does not decay at all. The implications of these spectral properties become evident in the training dynamics of neural networks. When using gradient descent, the evolution of the Fourier coefficients of $f(\mathbf{x})$ over time follows

$$\frac{d\hat{f}(\mathbf{k}, t)}{dt} = -\lambda(\mathbf{k})\hat{f}(\mathbf{k}, t), \tag{2.53}$$

where $\lambda(\mathbf{k})$ is an effective learning rate for each frequency. The decay behavior of $\hat{\sigma}(\mathbf{k})$ influences $\lambda(\mathbf{k})$, meaning that activation functions with strong spectral decay impose a bottleneck on the learning of high-frequency components. From a function approximation perspective, the spectral characteristics of the activation function determine the types of functions that can be efficiently represented by a neural network. Smooth activation functions lead to approximations that are predominantly low-frequency, whereas activation functions with slower Fourier decay allow the network to approximate functions with higher-frequency content. Thus, the activation function fundamentally shapes the Fourier spectrum of neural networks, controlling the network's ability to represent and learn different frequency components.

Activation Function	**Fourier Decay Rate**	**Effect on Frequency Learni**
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$	$e^{-\pi\|k\|}$ (Exponential)	Strong low-pass filter, retains only low f
Tanh $\sigma(x) = \tanh(x)$	$e^{-\pi\|k\|}$ (Exponential)	Strong low-pass filter, smooth approx
ReLU $\sigma(x) = \max(0, x)$	$\|k\|^{-2}$ (Power-law)	Allows moderate frequency learn
Leaky ReLU $\sigma(x) = \max(\alpha x, x)$	$\|k\|^{-2}$ (Power-law)	Similar to ReLU with slightly improved high-fr
Sinusoidal $\sigma(x) = \sin(x)$	No decay	Captures all frequencies, highly oscillato

This table summarizes the spectral characteristics of various activation functions and their impact on frequency learning in neural networks.

In summary, the Fourier analysis of expressivity rigorously demonstrates the superiority of deep networks over shallow ones in approximating complex functions. Depth introduces a hierarchical compositional structure that enables the efficient representation of high-frequency components, while width provides a rich basis for approximating the function's Fourier spectrum. Together, these properties explain the remarkable capacity of deep neural networks to approximate functions with intricate spectral structures, offering a mathematically rigorous foundation for understanding their expressivity.

2.2.3 Fourier transforms of various activation functions

To rigorously derive the Fourier transforms of various activation functions, let us consider the Fourier transform definition:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sigma(x) e^{-2\pi i k x} dx. \tag{2.54}$$

We will explicitly derive $\hat{\sigma}(k)$ for each activation function in a mathematically rigorous manner. This will include the Sigmoid, Tanh, ReLU, Leaky ReLU, and Sinusoidal activation functions.

Fourier Transform of the Sigmoid Function

The sigmoid activation function is given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.55}$$

The Fourier transform of $\sigma(x)$ is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{1}{1 + e^{-x}} e^{-2\pi i k x} dx. \tag{2.56}$$

Rewriting $\sigma(x)$ using the identity:

$$\sigma(x) = \frac{e^x}{1 + e^x}, \tag{2.57}$$

the Fourier integral becomes:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{e^x}{1+e^x} e^{-2\pi ikx} dx. \tag{2.58}$$

Setting $u = e^x$, so that $du = e^x dx$, we transform the integral:

$$\hat{\sigma}(k) = \int_0^{\infty} \frac{u}{1+u} u^{-2\pi ik} \frac{du}{u}. \tag{2.59}$$

Simplifying:

$$\hat{\sigma}(k) = \int_0^{\infty} \frac{u^{-2\pi ik+1}}{1+u} du. \tag{2.60}$$

This integral is well-known and can be computed using contour integration, yielding:

$$\hat{\sigma}(k) = \pi \text{csch}(\pi k), \tag{2.61}$$

where $\text{csch}(x) = \frac{1}{\sinh(x)}$ is the hyperbolic cosecant function. For large $|k|$, this decays as:

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.62}$$

Fourier Transform of the Hyperbolic Tangent Function

The tanh activation function is:

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.63}$$

The Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{e^x - e^{-x}}{e^x + e^{-x}} e^{-2\pi ikx} dx. \tag{2.64}$$

Using the identity:

$$\tanh(x) = 1 - 2\sigma(-x), \tag{2.65}$$

and leveraging the result from the sigmoid case, we obtain:

$$\hat{\sigma}(k) = -2\pi i \text{csch}(\pi k). \tag{2.66}$$

Again, for large $|k|$:

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.67}$$

Fourier Transform of the ReLU Function

The ReLU activation function is:

$$\sigma(x) = \max(0, x). \tag{2.68}$$

Thus, the Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} xH(x) e^{-2\pi ikx} dx, \tag{2.69}$$

where $H(x)$ is the Heaviside step function. Since $H(x)$ is zero for $x < 0$, the integral simplifies to:

$$\hat{\sigma}(k) = \int_0^{\infty} x e^{-2\pi ikx} dx. \tag{2.70}$$

Using integration by parts with $u = x$ and $dv = e^{-2\pi ikx} dx$, we find:

$$\hat{\sigma}(k) = \frac{1}{(2\pi ik)^2}. \tag{2.71}$$

This exhibits a power-law decay:

$$\hat{\sigma}(k) \sim |k|^{-2}. \tag{2.72}$$

Fourier Transform of the Leaky ReLU Function

The Leaky ReLU function is:

$$\sigma(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0. \end{cases} \tag{2.73}$$

The Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sigma(x) e^{-2\pi i k x} dx. \tag{2.74}$$

Splitting the integral:

$$\hat{\sigma}(k) = \int_{0}^{\infty} x e^{-2\pi i k x} dx + \alpha \int_{-\infty}^{0} x e^{-2\pi i k x} dx. \tag{2.75}$$

Computing these integrals using integration by parts gives:

$$\hat{\sigma}(k) = \frac{1+\alpha}{(2\pi i k)^2}. \tag{2.76}$$

Thus, the decay is still $|k|^{-2}$, but the amplitude is modulated by $(1+\alpha)$.

Fourier Transform of the Sinusoidal Activation Function

For $\sigma(x) = \sin(x)$, the Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sin(x) e^{-2\pi i k x} dx. \tag{2.77}$$

Using the Fourier transform properties of sine:

$$\hat{\sigma}(k) = \frac{i}{2} \left[\delta(k - \frac{1}{2\pi}) - \delta(k + \frac{1}{2\pi}) \right]. \tag{2.78}$$

This shows that the sinusoidal activation function contains only two specific frequencies and does not decay, meaning it retains high-frequency components perfectly.

These rigorous derivations confirm how different activation functions influence the Fourier spectrum of neural networks. The decay properties play a crucial role in determining the network's ability to learn high- or low-frequency functions.

3 Training Dynamics and NTK Linearization

Literature Review: Trevisan et. al. [85] investigated how knowledge distillation can be analyzed using the Neural Tangent Kernel (NTK) framework and demonstrated that under certain conditions, the training dynamics of a student model in knowledge distillation closely follow NTK linearization. They explored how NTK affects generalization and feature transfer in the distillation process. They provided theoretical insight into why knowledge distillation improves performance in deep networks. Bonfanti et. al. (2024) [86] studied how NTK behaves in the nonlinear regime, particularly in Physics-Informed Neural Networks (PINNs). They showed that when PINNs operate outside the NTK regime, their performance degrades due to high sensitivity to initialization and weight updates. They established conditions under which NTK linearization is insufficient for PINNs, emphasizing the need for nonlinear adaptations. They provided practical guidelines for designing PINNs that maintain stable training dynamics. Jacot et. al. (2018) [87] introduced the Neural Tangent Kernel (NTK) as a fundamental framework for analyzing infinite-width neural networks. They proved that as width approaches infinity, neural networks evolve as linear models governed by the NTK. They derived generalization bounds for infinitely wide networks and connected training dynamics to kernel methods. They established NTK as a core tool in deep learning theory, leading to further developments in training dynamics research. Lee et. al. (2019) [88] extended NTK theory to arbitrarily deep networks, showing that even deep architectures behave as linear models under gradient descent and proved that training dynamics remain stable regardless of network depth when width is sufficiently large. They explored practical implications for initializing and optimizing deep networks. They strengthened NTK theory by confirming its validity beyond shallow networks. Yang and Hu (2022) [89] challenged the conventional NTK assumption that feature learning is negligible in infinite-width networks and showed that certain activation functions can induce nontrivial feature learning even in infinite-width regimes. They suggested that feature learning can be integrated into NTK theory, opening new directions in kernel-based deep learning research. Xiang et. al. (2023) [90] investigated how finite-width effects impact training dynamics under NTK assumptions and showed that finite-width networks deviate from NTK predictions due to higher-order corrections in weight updates. They derived corrections to NTK theory for practical networks, improving its predictive power for real-world architectures. They refined NTK approximations, making them more applicable to modern deep-learning models. Lee et. al. (2019) [91] extended NTK linearization to deep convolutional networks, analyzing their training dynamics under infinite width and showed how locality and weight sharing in CNNs impact NTK behavior. They also demonstrated practical consequences for CNN training in real-world applications. They bridged NTK theory and convolutional architectures, providing new theoretical tools for CNN analysis.

3.1 Gradient Flow and Stationary Points

Literature Review: Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including a detailed discussion of gradient-based optimization methods. It rigorously explains the dynamics of gradient descent in the context of neural networks, covering topics such as backpropagation, vanishing gradients, and saddle points. The book also discusses the role of learning rates, momentum, and adaptive optimization methods in shaping the trajectory of gradient flow. Sra et. al. (2012) [474] included several chapters dedicated to the theoretical and practical aspects of gradient-based optimization in machine learning. It provides rigorous mathematical treatments of gradient flow dynamics, including convergence analysis, the impact of stochasticity in stochastic gradient descent (SGD), and the geometry of loss landscapes in high-dimensional spaces. Choromanska et. al. (2015) [475] rigorously analyzed the loss surfaces of deep neural networks. It demonstrates that the loss landscape is highly non-convex but contains a large number of local minima that are close in function value to the global minimum. The paper provides insights into how gradient flow navigates these complex landscapes and why it often converges to satisfactory solutions despite the non-convexity. Arora et al. (2019) [476] provided a theoretical framework for understanding the dynamics of gradient descent in deep neural networks. It rigorously analyzes the role of overparameterization in enabling gradient flow to converge to global minima, even in the absence of explicit regularization. The paper also explores the implicit regularization effects of gradient descent and their impact on generalization. Du et. al. (2019) [467] establishes theoretical guarantees for the convergence of gradient descent to global minima in overparameterized neural networks. It rigorously proves that gradient flow can efficiently minimize the training loss to zero, even in the presence of non-convexity, by leveraging the high-dimensional geometry of the loss landscape. The authors provided a rigorous analysis of the exponential convergence of gradient descent in overparameterized neural networks. It shows that the gradient flow dynamics are characterized by a rapid decrease in the loss function, driven by the alignment of the network's parameters with the data. The paper also discusses the role of initialization in shaping the trajectory of gradient flow. Zhang et al. (2017) [445] challenged traditional notions of generalization in deep learning. It rigorously demonstrates that deep neural networks can fit random labels, suggesting that the dynamics of gradient flow are not solely driven by the data distribution but also by the implicit biases of the optimization algorithm. The paper highlights the importance of understanding how gradient flow interacts with the architecture and initialization of neural networks. Baratin et. al. (2020) [477] explored the implicit regularization effects of gradient flow in deep learning from the perspective of function space. It rigorously demonstrates that gradient descent in overparameterized models tends to converge to solutions that minimize certain norms or complexity measures, providing insights into why these models generalize well despite their capacity to overfit. Balduzzi et al. (2018) [478] extended the analysis of gradient flow to multi-agent optimization problems, such as those encountered in generative adversarial networks (GANs). It rigorously characterizes the dynamics of gradient descent in games, highlighting the role of rotational forces and the challenges of convergence in non-cooperative settings. The paper provides tools for understanding how gradient flow behaves in complex, interactive learning scenarios. Allen-Zhu et al. (2019) [469] provided a rigorous convergence theory for deep learning models trained with gradient descent. It shows that overparameterization enables gradient flow to avoid bad local minima and converge to global minima efficiently. The paper also analyzes the role of initialization, step size, and network depth in shaping the dynamics of gradient descent.

The dynamics of gradient flow in neural network training are fundamentally governed by the continuous evolution of parameters $\theta(t)$ under the influence of the negative gradient of the loss function, expressed as

$$\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t)). \tag{3.1}$$

The loss function, typically of the form

$$\mathcal{L}(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \|f(x_i; \theta) - y_i\|^2, \tag{3.2}$$

measures the discrepancy between the network's predicted outputs $f(x_i;\theta)$ and the true labels y_i. At stationary points of the flow, the condition

$$\nabla_\theta \mathcal{L}(\theta^*) = 0 \tag{3.3}$$

holds, indicating that the gradient vanishes. To classify these stationary points, the Hessian matrix $H = \nabla_\theta^2 \mathcal{L}(\theta)$ is examined. For eigenvalues $\{\lambda_i\}$ of H, the nature of the stationary point is determined: $\lambda_i > 0$ for all i corresponds to a local minimum, $\lambda_i < 0$ for all i to a local maximum, and mixed signs indicate a saddle point.Under gradient flow $\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t))$, the trajectory converges to critical points:

$$\lim_{t\to\infty} \|\nabla_\theta \mathcal{L}(\theta(t))\| = 0. \tag{3.4}$$

The gradient flow also governs the temporal evolution of the network's predictions $f(x;\theta(t))$. A Taylor series expansion of $f(x;\theta)$ about an initial parameter θ_0 gives:

$$f(x;\theta) = f(x;\theta_0) + J_f(x;\theta_0)(\theta-\theta_0) + \frac{1}{2}(\theta-\theta_0)^\top H_f(x;\theta_0)(\theta-\theta_0) + \mathcal{O}(\|\theta-\theta_0\|^3), \tag{3.5}$$

where $J_f(x;\theta_0) = \nabla_\theta f(x;\theta_0)$ is the Jacobian and $H_f(x;\theta_0)$ is the Hessian of $f(x;\theta)$ with respect to θ. In the NTK (neural tangent kernel) regime, higher-order terms are negligible due to the large parameterization of the network, and the linear approximation suffices:

$$f(x;\theta) \approx f(x;\theta_0) + J_f(x;\theta_0)(\theta-\theta_0). \tag{3.6}$$

Under gradient flow, the time derivative of the network's predictions is given by:

$$\frac{df(x;\theta(t))}{dt} = J_f(x;\theta(t))\frac{d\theta(t)}{dt}. \tag{3.7}$$

Substituting the parameter dynamics $\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t)) = -\sum_{i=1}^n (f(x_i;\theta(t)) - y_i) J_f(x_i;\theta(t))$, this becomes:

$$\frac{df(x;\theta(t))}{dt} = -\sum_{i=1}^n J_f(x;\theta(t)) J_f(x_i;\theta(t))^\top (f(x_i;\theta(t)) - y_i). \tag{3.8}$$

Defining the NTK as $\mathcal{K}(x,x';\theta) = J_f(x;\theta)J_f(x';\theta)^\top$, and assuming constancy of the NTK during training ($\mathcal{K}(x,x';\theta) \approx \mathcal{K}_0(x,x')$), the evolution equation simplifies to:

$$\frac{df(x;\theta(t))}{dt} = -\sum_{i=1}^n \mathcal{K}_0(x,x_i)(f(x_i;\theta(t)) - y_i). \tag{3.9}$$

Rewriting in matrix form, let $\mathbf{f}(t) = [f(x_1;\theta(t)),\ldots,f(x_n;\theta(t))]^\top$ and $\mathbf{y} = [y_1,\ldots,y_n]^\top$. The NTK matrix $\mathcal{K}_0 \in \mathbb{R}^{n\times n}$ evaluated at initialization defines the system:

$$\frac{d\mathbf{f}(t)}{dt} = -\mathcal{K}_0(\mathbf{f}(t) - \mathbf{y}). \tag{3.10}$$

The solution to this linear system is:

$$\mathbf{f}(t) = e^{-\mathcal{K}_0 t}\mathbf{f}(0) + (\mathbf{I} - e^{-\mathcal{K}_0 t})\mathbf{y}. \tag{3.11}$$

As $t \to \infty$, the predictions converge to the labels: $\mathbf{f}(t) \to \mathbf{y}$, implying zero training error. The eigenvalues of $\mathcal{K}_0$ determine the rates of convergence. Diagonalizing $\mathcal{K}_0$ as $\mathcal{K}_0 = Q\Lambda Q^\top$, where Q is orthogonal and $\Lambda = \mathrm{diag}(\lambda_1,\ldots,\lambda_n)$, the dynamics in the eigenbasis are:

$$\frac{d\tilde{\mathbf{f}}(t)}{dt} = -\Lambda(\tilde{\mathbf{f}}(t) - \tilde{\mathbf{y}}), \tag{3.12}$$

with $\tilde{\mathbf{f}}(t) = Q^\top \mathbf{f}(t)$ and $\tilde{\mathbf{y}} = Q^\top \mathbf{y}$. Solving, we obtain:

$$\tilde{\mathbf{f}}(t) = e^{-\Lambda t}\tilde{\mathbf{f}}(0) + (\mathbf{I} - e^{-\Lambda t})\tilde{\mathbf{y}}. \tag{3.13}$$

Each mode decays exponentially with a rate proportional to the eigenvalue λ_i. Modes with larger λ_i converge faster, while smaller eigenvalues slow convergence.

The NTK framework thus rigorously explains the linearization of training dynamics in overparameterized neural networks. This linear behavior ensures that the optimization trajectory remains within a convex region of the parameter space, leading to both convergence and generalization. By leveraging the constancy of the NTK, the complexity of nonlinear neural networks is reduced to an analytically tractable framework that aligns closely with empirical observations.

3.1.1 Hessian Structure

The Hessian matrix, $H(\theta) = \nabla_\theta^2 \mathcal{L}(\theta)$, serves as a critical construct in the mathematical framework of optimization, capturing the second-order partial derivatives of the loss function $\mathcal{L}(\theta)$ with respect to the parameter vector $\theta \in \mathbb{R}^d$. Each element $H_{ij} = \frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta_i \partial\theta_j}$ reflects the curvature of the loss surface along the (i, j)-direction. The symmetry of $H(\theta)$, guaranteed by the Schwarz theorem under the assumption of continuous second partial derivatives, implies $H_{ij} = H_{ji}$. This property ensures that the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ of $H(\theta)$ are real and the eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ are orthogonal, satisfying the eigenvalue equation

$$H(\theta)\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \text{for all } i. \tag{3.14}$$

The behavior of the loss function around a specific parameter value θ_0 can be rigorously analyzed using a second-order Taylor expansion. This expansion is given by:

$$\mathcal{L}(\theta) = \mathcal{L}(\theta_0) + (\theta - \theta_0)^\top \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H(\theta_0)(\theta - \theta_0) + \mathcal{O}(\|\theta - \theta_0\|^3). \tag{3.15}$$

Here, the term $(\theta - \theta_0)^\top \nabla_\theta \mathcal{L}(\theta_0)$ represents the linear variation of the loss, while the quadratic term $\frac{1}{2}(\theta - \theta_0)^\top H(\theta_0)(\theta - \theta_0)$ describes the curvature effects. The eigenvalues of $H(\theta_0)$ dictate the nature of the critical point θ_0. Specifically, if all $\lambda_i > 0$, θ_0 is a local minimum; if all $\lambda_i < 0$, it is a local maximum; and if the eigenvalues have mixed signs, θ_0 is a saddle point. The leading-order approximation to the change in the loss function, $\Delta\mathcal{L} \approx \frac{1}{2}\delta\theta^\top H(\theta_0)\delta\theta$, highlights the dependence on the eigenstructure of $H(\theta_0)$. In the context of gradient descent, parameter updates follow the iterative scheme:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}(\theta^{(t)}), \tag{3.16}$$

where η is the learning rate. Substituting the Taylor expansion of $\nabla_\theta \mathcal{L}(\theta^{(t)})$ around θ_0 gives:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \left[\nabla_\theta \mathcal{L}(\theta_0) + H(\theta_0)(\theta^{(t)} - \theta_0)\right]. \tag{3.17}$$

To analyze this update rigorously, we project $\theta^{(t)} - \theta_0$ onto the eigenbasis of $H(\theta_0)$, expressing it as:

$$\theta^{(t)} - \theta_0 = \sum_{i=1}^{d} c_i^{(t)} \mathbf{v}_i, \tag{3.18}$$

where $c_i^{(t)} = \mathbf{v}_i^\top(\theta^{(t)} - \theta_0)$. Substituting this expansion into the gradient descent update rule yields:

$$c_i^{(t+1)} = c_i^{(t)} - \eta \left[\mathbf{v}_i^\top \nabla_\theta \mathcal{L}(\theta_0) + \lambda_i c_i^{(t)}\right]. \tag{3.19}$$

The convergence of this iterative scheme is governed by the condition $|1 - \eta\lambda_i| < 1$, which constrains the learning rate η relative to the spectrum of $H(\theta_0)$. For eigenvalues λ_i with large magnitudes, excessively large learning rates η can cause oscillatory or divergent updates.

In the Neural Tangent Kernel (NTK) regime, the evolution of a neural network during training can be approximated by a linearization of the network output around the initialization. Let $f_\theta(x)$ denote the output of the network for input x. Linearizing $f_\theta(x)$ around θ_0 gives:

$$f_\theta(x) \approx f_{\theta_0}(x) + \nabla_\theta f_{\theta_0}(x)^\top (\theta - \theta_0). \tag{3.20}$$

The NTK, defined as:

$$K(x, x') = \nabla_\theta f_{\theta_0}(x)^\top \nabla_\theta f_{\theta_0}(x'), \tag{3.21}$$

remains approximately constant during training for sufficiently wide networks. The training dynamics of the parameters are described by:

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta), \tag{3.22}$$

which, under the NTK approximation, becomes:

$$\frac{d\theta}{dt} = -K \nabla_\theta \mathcal{L}(\theta), \tag{3.23}$$

where K is the NTK matrix evaluated at initialization. The evolution of the loss function is governed by the eigenvalues of K, which control the rate of convergence in different directions.

The spectral properties of the Hessian play a pivotal role in the generalization properties of neural networks. Empirical studies reveal that the eigenvalue spectrum of $H(\theta)$ often exhibits a "bulk-and-spike" structure, with a dense bulk of eigenvalues near zero and a few large outliers. The bulk corresponds to flat directions in the loss landscape, which contribute to the robustness and generalization of the model, while the spikes represent sharp directions associated with overfitting. This spectral structure can be analyzed using random matrix theory, where the density of eigenvalues $\rho(\lambda)$ is modeled by distributions such as the Marchenko-Pastur law:

$$\rho(\lambda) = \frac{1}{2\pi\lambda q}\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}, \tag{3.24}$$

where $\lambda_\pm = (1 \pm \sqrt{q})^2$ are the spectral bounds and $q = \frac{d}{n}$ is the ratio of the number of parameters to the number of data points. This rigorous analysis links the Hessian structure to both the optimization dynamics and the generalization performance of neural networks, providing a comprehensive mathematical understanding of the training process. The Hessian $H(\theta)$ satisfies:

$$H(\theta) = \nabla_\theta^2 \mathcal{L}(\theta) = \mathbb{E}_{(x,y)}\left[\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^\top\right]. \tag{3.25}$$

For overparameterized networks, $H(\theta)$ is nearly degenerate, implying the existence of flat minima.

3.1.2 NTK Linearization

The dynamics of neural networks under gradient flow can be comprehensively described by beginning with the parameterized representation of the network $f_\theta(x)$, where $\theta \in \mathbb{R}^p$ denotes the set of trainable parameters, $x \in \mathbb{R}^d$ is the input, and $f_\theta(x) \in \mathbb{R}^m$ represents the output. The objective of training is to minimize a loss function $\mathcal{L}(\theta)$, defined over a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^m$ represent the input-target pairs. The evolution of the parameters during training is governed by the gradient flow equation $\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta)$, where $\nabla_\theta \mathcal{L}(\theta)$ is the gradient of the loss function with respect to the parameters. To analyze the dynamics of the network outputs, we first consider the time derivative of $f_\theta(x)$. Using the chain rule, this is expressed as:

$$\frac{\partial f_\theta(x)}{\partial t} = \nabla_\theta f_\theta(x)^\top \frac{d\theta}{dt}. \tag{3.26}$$

Substituting $\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta)$, we have:

$$\frac{\partial f_\theta(x)}{\partial t} = -\nabla_\theta f_\theta(x)^\top \nabla_\theta \mathcal{L}(\theta). \tag{3.27}$$

The gradient of the loss function, $\mathcal{L}(\theta)$, can be expressed explicitly in terms of the training data. For a generic loss function over the dataset, this takes the form:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i), \tag{3.28}$$

where $\ell(f_\theta(x_i), y_i)$ represents the loss for the i-th data point. The gradient of the loss with respect to the parameters is therefore given by:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f_\theta(x_i) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.29}$$

Substituting this back into the time derivative of $f_\theta(x)$, we obtain:

$$\frac{\partial f_\theta(x)}{\partial t} = -\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f_\theta(x)^\top \nabla_\theta f_\theta(x_i) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.30}$$

To introduce the Neural Tangent Kernel (NTK), we define it as the Gram matrix of the Jacobians of the network output with respect to the parameters:

$$\Theta(x, x'; \theta) = \nabla_\theta f_\theta(x)^\top \nabla_\theta f_\theta(x'). \tag{3.31}$$

Using this definition, the time evolution of the output becomes:

$$\frac{\partial f_\theta(x)}{\partial t} = -\frac{1}{n}\sum_{i=1}^{n} \Theta(x, x_i; \theta) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.32}$$

In the overparameterized regime, where the number of parameters p is significantly larger than the number of training data points n, it has been empirically and theoretically observed that the NTK $\Theta(x, x'; \theta)$ remains nearly constant during training. Specifically, $\Theta(x, x'; \theta) \approx \Theta(x, x'; \theta_0)$, where θ_0 represents the parameters at initialization. This constancy significantly simplifies the analysis of the network's training dynamics. To see this, consider the solution to the differential equation governing the output dynamics. Let $F(t) \in \mathbb{R}^{n\times m}$ represent the matrix of network outputs for all training inputs, where the i-th row corresponds to $f_\theta(x_i)$. The dynamics can be expressed in matrix form as:

$$\frac{\partial F(t)}{\partial t} = -\frac{1}{n}\Theta(\theta_0)\nabla_F \mathcal{L}(F), \tag{3.33}$$

where $\Theta(\theta_0) \in \mathbb{R}^{n\times n}$ is the NTK matrix evaluated at initialization, and $\nabla_F \mathcal{L}(F)$ is the gradient of the loss with respect to the output matrix F. For the special case of a mean squared error loss, $\mathcal{L}(F) = \frac{1}{2n}\|F - Y\|_F^2$, where $Y \in \mathbb{R}^{n\times m}$ is the matrix of target outputs, the gradient simplifies to:

$$\nabla_F \mathcal{L}(F) = \frac{1}{n}(F - Y). \tag{3.34}$$

Substituting this into the dynamics, we obtain:

$$\frac{\partial F(t)}{\partial t} = -\frac{1}{n^2}\Theta(\theta_0)(F(t) - Y). \tag{3.35}$$

The solution to this differential equation is:

$$F(t) = Y + e^{-\frac{\Theta(\theta_0)}{n^2}t}(F(0) - Y), \tag{3.36}$$

where $F(0)$ represents the initial outputs of the network. As $t \to \infty$, the exponential term vanishes, and the network outputs converge to the targets Y, provided that $\Theta(\theta_0)$ is positive definite. The rate of convergence is determined by the eigenvalues of $\Theta(\theta_0)$, with smaller eigenvalues corresponding to slower convergence along the associated eigenvectors. To understand the stationary points of this system, we note that these occur when $\frac{\partial F(t)}{\partial t} = 0$. From the dynamics, this implies:

$$\Theta(\theta_0)(F - Y) = 0. \tag{3.37}$$

If $\Theta(\theta_0)$ is invertible, this yields $F = Y$, indicating that the network exactly interpolates the training data at the stationary point. However, if $\Theta(\theta_0)$ is not full-rank, the stationary points form a subspace of solutions satisfying $(I - \Pi)(F - Y) = 0$, where Π is the projection operator onto the column space of $\Theta(\theta_0)$.

The NTK framework provides a mathematically rigorous lens to analyze training dynamics, elucidating the interplay between parameter evolution, kernel properties, and loss convergence in neural networks. By linearizing the training dynamics through the NTK, we achieve a deep understanding of how overparameterized networks evolve under gradient flow and how they reach stationary points, revealing their capacity to interpolate data with remarkable precision.

3.2 NTK Regime

Literature Review: Jacot et. al. (2018) [87] in a seminal paper introduced the Neural Tangent Kernel (NTK) and establishes its theoretical foundation. The authors show that in the infinite-width limit, the dynamics of gradient descent in neural networks can be described by a kernel method, where the NTK remains constant during training. This work bridges the gap between deep learning and kernel methods, providing a framework to analyze the training and generalization of wide neural networks. Lee et. al. (2017) [88] did a work that predates the NTK but lays the groundwork by showing that infinitely wide neural networks behave as Gaussian processes. The authors derive the kernel corresponding to such networks, which is a precursor to the NTK. This paper is crucial for understanding the connection between neural networks and kernel methods. Chizat and Bach (2018) [466] provided a rigorous analysis of gradient descent in over-parameterized models, including neural networks. It complements the NTK framework by showing that gradient descent converges to global minima in such settings. The work highlights the role of over-parameterization in simplifying the optimization landscape. Du et. al. (2019) [467] proved that gradient descent can find global minima in deep neural networks under the NTK regime. The authors provide explicit convergence rates and show that the NTK framework guarantees efficient optimization for wide networks. This work strengthens the theoretical understanding of why deep learning works. Arora et. al. (2019) [468] provided a fine-grained analysis of optimization and generalization in two-layer neural networks under the NTK regime. It establishes precise bounds on the generalization error and shows how over-parameterization leads to benign optimization landscapes. Allen-Zhu et. al. (2019) [469] extended the NTK framework to deep networks and provided a comprehensive convergence theory for over-parameterized neural networks. The authors show that gradient descent converges to global minima and that the NTK remains approximately constant during training. Cao and Gu (2019) [470] derived generalization bounds for wide and deep neural networks trained with stochastic gradient descent (SGD) under the NTK regime. It highlights the role of the NTK in controlling the generalization error and provides insights into the implicit regularization of SGD. Yang (2019) [471] generalized the NTK framework to architectures with weight sharing, such as convolutional neural networks (CNNs). The author derives the NTK for such architectures and shows that they also exhibit Gaussian process behavior in the infinite-width limit. Huang and Yau (2020) [472] extended the NTK framework by introducing the Neural Tangent Hierarchy (NTH), which captures higher-order interactions in the training dynamics of deep networks. The authors provide a more refined analysis of the training process beyond the first-order approximation of the NTK. Belkin et. al. (2019) [473] explored the connection between deep learning and kernel learning, emphasizing the role of the NTK in understanding generalization and optimization. It provides a high-level perspective on why the NTK framework is essential for analyzing modern machine learning models.

The Neural Tangent Kernel (NTK) regime is a fundamental framework for understanding the dynamics of gradient descent in highly overparameterized neural networks. Consider a neural network $f(\mathbf{x};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^P$, where P represents the total number of parameters, and $\mathbf{x} \in \mathbb{R}^d$ is the input vector. For a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the loss function $L(t)$ at time t is given by

$$L(t) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i;\boldsymbol{\theta}(t)) - y_i\right)^2. \tag{3.38}$$

The parameters evolve according to gradient descent as $\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta\nabla_{\boldsymbol{\theta}}L(t)$, where $\eta > 0$ is the learning rate. In the NTK regime, we consider the first-order Taylor expansion of the network output around the initialization $\boldsymbol{\theta}_0$:

$$f(\mathbf{x};\boldsymbol{\theta}) \approx f(\mathbf{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}}f(\mathbf{x};\boldsymbol{\theta}_0)^\top(\boldsymbol{\theta} - \boldsymbol{\theta}_0). \tag{3.39}$$

This linear approximation transforms the nonlinear dynamics of f into a simpler, linearized form. To analyze training, we introduce the Jacobian matrix $J \in \mathbb{R}^{N\times P}$, where $J_{ij} = \frac{\partial f(\mathbf{x}_i;\boldsymbol{\theta}_0)}{\partial\theta_j}$. The vector of outputs $\mathbf{f}(t) \in \mathbb{R}^N$, aggregating predictions over the dataset, evolves as

$$\mathbf{f}(t) = \mathbf{f}(0) + J(\boldsymbol{\theta}(t) - \boldsymbol{\theta}_0). \tag{3.40}$$

The NTK $\Theta \in \mathbb{R}^{N\times N}$ is defined as

$$\Theta_{ij} = \nabla_{\boldsymbol{\theta}}f(\mathbf{x}_i;\boldsymbol{\theta}_0)^\top\nabla_{\boldsymbol{\theta}}f(\mathbf{x}_j;\boldsymbol{\theta}_0). \tag{3.41}$$

As $P \to \infty$, the NTK converges to a deterministic matrix that remains nearly constant during training. Substituting the linearized form of $\mathbf{f}(t)$ into the gradient descent update equation gives

$$\mathbf{f}(t+1) = \mathbf{f}(t) - \frac{\eta}{N}\Theta(\mathbf{f}(t) - \mathbf{y}), \tag{3.42}$$

where $\mathbf{y} \in \mathbb{R}^N$ is the vector of true labels. Defining the residual $\mathbf{r}(t) = \mathbf{f}(t) - \mathbf{y}$, the dynamics of training reduce to

$$\mathbf{r}(t+1) = \left(I - \frac{\eta}{N}\Theta\right)\mathbf{r}(t). \tag{3.43}$$

The eigendecomposition $\Theta = Q\Lambda Q^\top$, with orthogonal Q and diagonal $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$, allows us to analyze the decay of residuals in the eigenbasis of Θ:

$$\tilde{\mathbf{r}}(t+1) = \left(I - \frac{\eta}{N}\Lambda\right)\tilde{\mathbf{r}}(t), \tag{3.44}$$

where $\tilde{\mathbf{r}}(t) = Q^\top\mathbf{r}(t)$. Each component decays as

$$\tilde{r}_i(t) = \left(1 - \frac{\eta\lambda_i}{N}\right)^t \tilde{r}_i(0). \tag{3.45}$$

For small η, the training dynamics are approximately continuous, governed by

$$\frac{d\mathbf{r}(t)}{dt} = -\frac{1}{N}\Theta\mathbf{r}(t), \tag{3.46}$$

leading to the solution

$$\mathbf{r}(t) = \exp\left(-\frac{\Theta t}{N}\right)\mathbf{r}(0). \tag{3.47}$$

The NTK for specific architectures, such as fully connected ReLU networks, can be derived using layerwise covariance matrices. Let $\Sigma^{(l)}(\mathbf{x}, \mathbf{x}')$ denote the covariance between pre-activations at layer l. The recurrence relation for $\Sigma^{(l)}$ is

$$\Sigma^{(l)}(\mathbf{x}, \mathbf{x}') = \frac{1}{2\pi} \|\mathbf{z}^{(l-1)}(\mathbf{x})\| \|\mathbf{z}^{(l-1)}(\mathbf{x}')\| \left(\sin\theta + (\pi - \theta)\cos\theta\right), \tag{3.48}$$

where $\theta = \cos^{-1}\left(\frac{\Sigma^{(l-1)}(\mathbf{x},\mathbf{x}')}{\sqrt{\Sigma^{(l-1)}(\mathbf{x},\mathbf{x})\Sigma^{(l-1)}(\mathbf{x}',\mathbf{x}')}}\right)$. The NTK, a sum over contributions from all layers, quantifies how parameter updates propagate through the network.

In the infinite-width limit, the NTK framework predicts generalization properties, as the kernel matrix Θ governs both training and test-time behavior. The NTK connects neural networks to classical kernel methods, offering a bridge between deep learning and well-established theoretical tools in approximation theory. This regime's deterministic and analytical tractability enables precise characterizations of network performance, convergence rates, and robustness to initialization and learning rate variations.

4 Generalization Bounds: PAC-Bayes and Spectral Analysis

4.1 PAC-Bayes Formalism

Literature Review: McAllester (1999) [92] introduced the PAC-Bayes bound, a fundamental theorem that provides generalization guarantees for Bayesian learning models. He established a trade-off between complexity and empirical risk, serving as the theoretical foundation for modern PAC-Bayesian analysis. Catoni (2007) [93] in his book rigorously extended the PAC-Bayes framework by linking it with information-theoretic and statistical mechanics concepts and introduced exponential and Gibbs priors for learning, improving PAC-Bayesian bounds for supervised classification. Germain et. al. (2009) [94] applied PAC-Bayes theory to linear classifiers, including SVMs and logistic regression. They demonstrated that PAC-Bayesian generalization bounds are tighter than classical Vapnik-Chervonenkis (VC) dimension bounds. Seeger (2002) [95] extended PAC-Bayes bounds to Gaussian Process models, proving tight generalization guarantees for Bayesian classifiers. He laid the groundwork for probabilistic kernel methods. Alquier et. al. (2006) [96] connected variational inference and PAC-Bayes bounds, proving that variational approximations can preserve generalization guarantees of PAC-Bayesian bounds. Dziugaite and Roy (2017) [97] gave one of the first applications of PAC-Bayes to deep learning. They derived nonvacuous generalization bounds for stochastic neural networks, bridging theory and practice. Rivasplata et. al. (2020) [98] provided novel PAC-Bayes bounds that improve over existing guarantees, making PAC-Bayesian bounds more practical for modern ML applications. Lever et. al. (2013) [99] explored data-dependent priors in PAC-Bayes theory, showing that adaptive priors lead to tighter generalization bounds. Rivasplata et. al. (2018) [100] introduced instance-dependent priors, improving personalized learning and making PAC-Bayesian methods more useful for real-world machine learning problems. Lindemann et. al. (2024) [101] integrated PAC-Bayes theory with conformal prediction to improve formal verification in control systems, demonstrating PAC-Bayes' relevance to safety-critical applications.

The PAC-Bayes formalism is a foundational framework in statistical learning theory, designed to provide probabilistic guarantees on the generalization performance of learning algorithms. By combining principles from the PAC (Probably Approximately Correct) framework and Bayesian reasoning, PAC-Bayes delivers bounds that characterize the expected performance of hypotheses drawn from posterior distributions, given a finite sample of data. This document presents an extremely rigorous and mathematically precise description of the PAC-Bayes formalism, emphasizing its theoretical constructs and implications.

At the core of the PAC-Bayes formalism lies the ambition to rigorously quantify the generalization ability of hypotheses $h \in \mathcal{H}$ based on their performance on a finite dataset $S \sim \mathcal{D}^m$, where $\mathcal{D}$ represents the underlying, and typically unknown, data distribution. The PAC framework, which was originally designed to provide high-confidence guarantees on the true risk

$$R(h) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)], \tag{4.1}$$

is enriched in PAC-Bayes by incorporating principles from Bayesian reasoning. This integration allows for bounds not just on individual hypotheses but on distributions Q over $\mathcal{H}$, yielding a sophisticated characterization of generalization that inherently accounts for the variability and uncertainty in the hypothesis space. There are some Mathematical Constructs: True and Empirical Risks. The true risk $R(h)$, as defined by the expected loss, is typically inaccessible due to the unknown nature of $\mathcal{D}$. Instead, the empirical risk

$$\hat{R}(h,S) = \frac{1}{m}\sum_{i=1}^{m} \ell(h,z_i) \tag{4.2}$$

serves as a computable proxy. The key question addressed by PAC-Bayes is: *How does $\hat{R}(h,S)$ relate to $R(h)$, and how can we bound the deviation probabilistically?* For a distribution Q over $\mathcal{H}$, these risks are generalized as:

$$R(Q) = \mathbb{E}_{h\sim Q}[R(h)], \quad \hat{R}(Q,S) = \mathbb{E}_{h\sim Q}[\hat{R}(h,S)]. \tag{4.3}$$

This generalization is pivotal because it allows the analysis to transcend individual hypotheses and consider probabilistic ensembles, where $Q(h)$ represents a posterior belief over the hypothesis space conditioned on the observed data. We now need to discuss how Prior and Posterior Distributions encode knowledge and complexity. The prior P is a fixed distribution over $\mathcal{H}$ that reflects pre-data assumptions about the plausibility of hypotheses. Crucially, P must be independent of S to avoid biasing the bounds. The posterior Q, however, is data-dependent and typically chosen to minimize a combination of empirical risk and complexity. This choice is guided by the PAC-Bayes inequality, which regularizes Q via its Kullback-Leibler (KL) divergence from P:

$$\mathrm{KL}(Q\|P) = \int_{\mathcal{H}} Q(h)\log\frac{Q(h)}{P(h)}\,dh. \tag{4.4}$$

The KL divergence quantifies the informational cost of updating P to Q, serving as a penalty term that discourages overly complex posteriors. This regularization is critical in preventing overfitting, ensuring that Q achieves a balance between data fidelity and model simplicity.

Let's derive the PAC-Bayes Inequality: Probabilistic and Information-Theoretic Foundations. The derivation of the PAC-Bayes inequality hinges on a combination of probabilistic tools and information-theoretic arguments. A central step involves applying a change of measure from P to Q, leveraging the identity:

$$\mathbb{E}_{h\sim Q}[f(h)] = \mathbb{E}_{h\sim P}\left[f(h)\frac{Q(h)}{P(h)}\right]. \tag{4.5}$$

This allows the incorporation of Q into bounds that originally apply to fixed h. By analyzing the moment-generating function of deviations between $\hat{R}(h,S)$ and $R(h)$, and applying Hoeffding's inequality to the empirical loss, we arrive at the following bound for any Q and P, with probability at least $1-\delta$:

$$R(Q) \le \hat{R}(Q,S) + \sqrt{\frac{\mathrm{KL}(Q\|P) + \log\frac{1}{\delta}}{2m}}. \tag{4.6}$$

The generalization bound is therefore given by:

$$\mathcal{L}(f) - \mathcal{L}_{\text{emp}}(f) \le \sqrt{\frac{\mathcal{KL}(Q\|P) + \log(1/\delta)}{2N}}, \tag{4.7}$$

where $\mathcal{KL}(Q\|P)$ quantifies the divergence between the posterior Q and prior P. This bound is remarkable because it explicitly ties the true risk $R(Q)$ to the empirical risk $\hat{R}(Q,S)$, the KL divergence, and the sample size m. The PAC-Bayes bound encapsulates three competing forces: the empirical risk $\hat{R}(Q,S)$, the complexity penalty $\mathrm{KL}(Q\|P)$, and the confidence term $\sqrt{\frac{\log\frac{1}{\delta}}{2m}}$. This interplay reflects a fundamental trade-off in learning:

1. **Empirical Risk:** $\hat{R}(Q, S)$ captures how well the posterior Q fits the training data.
2. **Complexity:** The KL divergence ensures that Q remains close to P, discouraging overfitting and promoting generalization.
3. **Confidence:** The term $\sqrt{\frac{\log \frac{1}{\delta}}{2m}}$ shrinks with increasing sample size, tightening the bound and enhancing reliability.

The KL term also introduces an inherent regularization effect, penalizing hypotheses that deviate significantly from prior knowledge. This aligns with Occam's Razor, favoring simpler explanations that are consistent with the data.

There are several extensions and Advanced Applications of Pac-Bayes Formalism. While the classical PAC-Bayes framework assumes i.i.d. data, recent advancements have generalized the theory to handle structured data, such as in time-series and graph-based learning. Furthermore, alternative divergence measures, like Rényi divergence or Wasserstein distance, have been explored to accommodate scenarios where KL divergence may be inappropriate. In practical settings, PAC-Bayes bounds have been instrumental in analyzing neural networks, Bayesian ensembles, and stochastic processes, offering theoretical guarantees even in high-dimensional, non-convex optimization landscapes.

4.1.1 KL divergence

The Kullback-Leibler (KL) divergence, also known as the relative entropy, is a fundamental concept in information theory, probability theory, and statistics. It quantifies the difference between two probability distributions, measuring the inefficiency of assuming that data is generated from one distribution when it is actually generated from another. Mathematically, for two discrete probability distributions $P(x)$ and $Q(x)$ defined over a common sample space $\mathcal{X}$, the KL divergence is given by

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{4.8}$$

where the logarithm is conventionally taken to be the natural logarithm unless otherwise specified. For continuous probability distributions with probability density functions $p(x)$ and $q(x)$, the KL divergence is defined as

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \, dx. \tag{4.9}$$

The KL divergence can be understood as the expectation of the logarithmic difference between the two distributions, weighted by the true distribution $P(x)$:

$$D_{\mathrm{KL}}(P \parallel Q) = \mathbb{E}_P \left[\log \frac{P(x)}{Q(x)} \right]. \tag{4.10}$$

Expanding this expectation in integral form,

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log p(x) \, dx - \int_{\mathcal{X}} p(x) \log q(x) \, dx. \tag{4.11}$$

The first integral represents the entropy of $P(x)$, denoted as

$$H(P) = -\int_{\mathcal{X}} p(x) \log p(x) \, dx, \tag{4.12}$$

while the second integral represents the cross-entropy between $P(x)$ and $Q(x)$, given by

$$H(P, Q) = -\int_{\mathcal{X}} p(x) \log q(x) \, dx. \tag{4.13}$$

Thus, KL divergence can be rewritten as

$$D_{\mathrm{KL}}(P \parallel Q) = H(P, Q) - H(P), \tag{4.14}$$

which represents the additional number of bits required to encode data from $P(x)$ when using a coding scheme optimized for $Q(x)$ instead of $P(x)$. Since the logarithm function is concave, an application of Jensen's inequality to the function $f(x) = -\log x$ yields the fundamental property

$$D_{\mathrm{KL}}(P \parallel Q) \geq 0, \tag{4.15}$$

with equality if and only if $P(x) = Q(x)$ for all x, which follows from the strict convexity of the logarithm function. Unlike a metric, KL divergence is not symmetric, meaning that in general,

$$D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P). \tag{4.16}$$

For two multivariate normal distributions $P = \mathcal{N}(\mu_1, \Sigma_1)$ and $Q = \mathcal{N}(\mu_2, \Sigma_2)$, the KL divergence is given by

$$D_{\mathrm{KL}}(P \parallel Q) = \frac{1}{2}\left[\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) - k + \log \frac{\det \Sigma_2}{\det \Sigma_1}\right], \tag{4.17}$$

where k is the dimension of the Gaussian distribution. This expression reveals that KL divergence accounts for both the difference in mean vectors and the difference in covariance structures. The KL divergence has a fundamental connection to maximum likelihood estimation (MLE). Given a true data distribution $P(x)$ and a parametric model $Q(x \mid \theta)$, the parameter θ that minimizes the KL divergence

$$D_{\mathrm{KL}}(P \parallel Q_\theta) = \mathbb{E}_P\left[\log P(x) - \log Q(x \mid \theta)\right] \tag{4.18}$$

is equivalent to the maximum likelihood estimate, since minimizing KL divergence is equivalent to maximizing the log-likelihood function

$$\sum_{i=1}^{N} \log Q(x_i \mid \theta). \tag{4.19}$$

This follows from the fact that the term $\mathbb{E}_P[\log P(x)]$ does not depend on θ and is therefore irrelevant for optimization. In Bayesian inference, KL divergence plays a central role in variational inference. Given an intractable posterior $P(z \mid x)$, one approximates it with a tractable distribution $Q(z)$, where the objective is to minimize

$$D_{\mathrm{KL}}(Q(z) \parallel P(z \mid x)). \tag{4.20}$$

This minimization leads to the Evidence Lower Bound (ELBO),

$$\log P(x) \geq \mathbb{E}_Q[\log P(x, z)] - \mathbb{E}_Q[\log Q(z)], \tag{4.21}$$

which is central in modern probabilistic modeling. Moreover, KL divergence is related to mutual information. Given two random variables X and Y with a joint distribution $P(X, Y)$ and marginals $P(X)$ and $P(Y)$, the mutual information is given by

$$I(X; Y) = D_{\mathrm{KL}}(P(X, Y) \parallel P(X)P(Y)), \tag{4.22}$$

which quantifies the amount of information shared between X and Y. In information geometry, the Fisher information matrix is derived from the second-order expansion of KL divergence,

$$g_{ij} = \mathbb{E}_P\left[\frac{\partial \log P(x \mid \theta)}{\partial \theta^i} \frac{\partial \log P(x \mid \theta)}{\partial \theta^j}\right]. \tag{4.23}$$

KL divergence is also connected to the Pinsker inequality,

$$\|P - Q\|_{\mathrm{TV}} \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}(P \parallel Q)}. \tag{4.24}$$

This establishes KL divergence as a fundamental measure of probability distance in statistical inference and machine learning.

4.1.2 Rényi divergence

The Rényi divergence is a parametric family of divergences that generalizes the Kullback-Leibler (KL) divergence and provides a broader measure of dissimilarity between probability distributions. Given two probability distributions P and Q defined on a measurable space $(\Omega, \mathcal{F})$, the Rényi divergence of order α (for $\alpha > 0$ and $\alpha \neq 1$) is defined as

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \int_\Omega p(x)^\alpha q(x)^{1-\alpha} \, d\mu(x), \tag{4.25}$$

where $p(x)$ and $q(x)$ denote the Radon-Nikodym derivatives (i.e., probability density functions) of P and Q with respect to a common reference measure μ. That is,

$$p(x) = \frac{dP}{d\mu}, \quad q(x) = \frac{dQ}{d\mu}. \tag{4.26}$$

For the case where P and Q are discrete distributions supported on a finite or countable set $\mathcal{X}$, the Rényi divergence takes the form

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \sum_{x \in \mathcal{X}} p(x)^\alpha q(x)^{1-\alpha}. \tag{4.27}$$

Equivalently, it can be expressed using expectation notation as

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \mathbb{E}_Q \left[\left(\frac{p(X)}{q(X)} \right)^\alpha \right]. \tag{4.28}$$

This expression shows that Rényi divergence is a moment-based measure of dissimilarity between P and Q, generalizing the standard KL divergence, which is recovered in the limit $\alpha \to 1$. To see this, we differentiate

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \int_\Omega p(x)^\alpha q(x)^{1-\alpha} \, d\mu(x), \tag{4.29}$$

with respect to α and apply L'Hôpital's rule. Explicitly, defining

$$F(\alpha) = \int_\Omega p(x)^\alpha q(x)^{1-\alpha} \, d\mu(x), \tag{4.30}$$

we compute

$$\frac{d}{d\alpha} F(\alpha) = \int_\Omega p(x)^\alpha q(x)^{1-\alpha} \ln \frac{p(x)}{q(x)} \, d\mu(x). \tag{4.31}$$

Applying the logarithmic derivative, we obtain

$$\lim_{\alpha \to 1} D_\alpha(P\|Q) = \sum_{x \in \mathcal{X}} p(x) \ln \frac{p(x)}{q(x)} = D_{\mathrm{KL}}(P\|Q), \tag{4.32}$$

demonstrating the convergence to the KL divergence. The Rényi divergence satisfies the following fundamental mathematical properties. It is non-negative,

$$D_\alpha(P\|Q) \geq 0, \quad \text{with equality if and only if } P = Q. \tag{4.33}$$

Additionally, Rényi divergence is a monotonic function of α, satisfying

$$\alpha_1 < \alpha_2 \Rightarrow D_{\alpha_1}(P\|Q) \leq D_{\alpha_2}(P\|Q). \tag{4.34}$$

For special values of α, Rényi divergence simplifies to various known divergences. When $\alpha = 0$, it becomes

$$D_0(P\|Q) = -\ln \sum_{\{x : p(x) > 0\}} q(x), \tag{4.35}$$

which represents the negative logarithm of the probability mass that Q assigns to the support of P. When $\alpha = 2$, we obtain

$$D_2(P\|Q) = \ln \sum_{x\in\mathcal{X}} \frac{p(x)^2}{q(x)}, \tag{4.36}$$

which is related to the chi-squared divergence. When $\alpha = \infty$, the divergence simplifies to

$$D_\infty(P\|Q) = \ln \sup_{x\in\mathcal{X}} \frac{p(x)}{q(x)}, \tag{4.37}$$

which measures the worst-case discrepancy between the two distributions. One crucial property of the Rényi divergence is its invariance under transformations. Given a Markov kernel T that maps P and Q to new distributions TP and TQ, the data-processing inequality states

$$D_\alpha(TP\|TQ) \leq D_\alpha(P\|Q), \tag{4.38}$$

ensuring that no transformation can increase the divergence between distributions. Rényi divergence also satisfies the joint convexity property: for a mixture of probability distributions P_i and Q_i with weights λ_i,

$$D_\alpha\left(\sum_i \lambda_i P_i \middle\| \sum_i \lambda_i Q_i\right) \leq \sum_i \lambda_i D_\alpha(P_i\|Q_i). \tag{4.39}$$

This inequality demonstrates that Rényi divergence behaves well under probabilistic mixing. Another fundamental result is the Pinsker-type bound, which states that for $\alpha > 1$, the Rényi divergence is bounded below by a function of the total variation distance $d_{\mathrm{TV}}(P, Q)$,

$$D_\alpha(P\|Q) \geq \frac{\alpha}{\alpha - 1} \ln\left(1 + \frac{\alpha - 1}{\alpha} d_{\mathrm{TV}}(P, Q)^2\right). \tag{4.40}$$

For small $d_{\mathrm{TV}}(P, Q)$, this inequality shows that Rényi divergence behaves quadratically in the deviation between P and Q. Furthermore, the Rényi divergence is closely related to the Chernoff information, which determines the optimal exponent for Bayesian hypothesis testing. The Chernoff information is defined as

$$C(P, Q) = -\min_{\alpha\in(0,1)} D_\alpha(P\|Q), \tag{4.41}$$

which plays a fundamental role in large deviations theory and hypothesis testing.

In summary, the Rényi divergence is a highly versatile and mathematically rich measure of dissimilarity between probability distributions. It generalizes the KL divergence, interpolates between various known divergence measures, satisfies essential properties such as monotonicity and data-processing inequalities, and has deep connections to large deviation theory, statistical estimation, and hypothesis testing.

4.1.3 Wasserstein distance

The Wasserstein distance, also referred to as the Earth Mover's Distance (EMD), is a fundamental metric in the field of optimal transport theory that rigorously quantifies the distance between two probability measures by determining the minimal cost required to transport one measure into another. Consider two probability distributions μ and ν, defined on a metric space $(\mathcal{X}, d)$. The Wasserstein distance of order $p \geq 1$ is defined by

$$W_p(\mu, \nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} d(x, y)^p \, d\gamma(x, y)\right)^{\frac{1}{p}} \tag{4.42}$$

where $\Pi(\mu,\nu)$ denotes the set of all probability measures γ on the product space $\mathcal{X}\times\mathcal{X}$ satisfying

$$\int_{\mathcal{X}} d\gamma(x,y)=\mu(x), \quad \int_{\mathcal{X}} d\gamma(x,y)=\nu(y). \tag{4.43}$$

This definition arises from the optimal transport problem, originally formulated by Monge in 1781 and later extended by Kantorovich. The Monge formulation seeks a transport map $T:\mathcal{X}\to\mathcal{X}$ that pushes μ onto ν, i.e., satisfies

$$T_{\#}\mu=\nu \tag{4.44}$$

where $T_{\#}\mu$ denotes the pushforward measure defined by

$$(T_{\#}\mu)(A)=\mu(T^{-1}(A)), \quad \forall A\subseteq\mathcal{X}. \tag{4.45}$$

The Monge formulation minimizes the total transport cost given by

$$\inf_{T:T_{\#}\mu=\nu}\int_{\mathcal{X}} d(x,T(x))^p d\mu(x). \tag{4.46}$$

However, the existence of an optimal transport map is highly constrained and may fail in many cases. To overcome this, Kantorovich introduced a relaxation in which transport maps are replaced by probability couplings γ, leading to the variational problem

$$W_p^p(\mu,\nu)=\inf_{\gamma\in\Pi(\mu,\nu)}\int_{\mathcal{X}\times\mathcal{X}} d(x,y)^p d\gamma(x,y). \tag{4.47}$$

This formulation allows for mass splitting, making the problem more tractable and ensuring the existence of minimizers under mild conditions. The dual formulation of the Wasserstein distance is given by

$$W_p^p(\mu,\nu)=\sup_{\varphi,\psi}\left\{\int_{\mathcal{X}}\varphi(x)d\mu(x)+\int_{\mathcal{X}}\psi(y)d\nu(y)\right\} \tag{4.48}$$

subject to the constraint

$$\varphi(x)+\psi(y)\le d(x,y)^p, \quad \forall x,y\in\mathcal{X}. \tag{4.49}$$

For $p=1$, the Kantorovich-Rubinstein theorem provides an alternative characterization

$$W_1(\mu,\nu)=\sup_{\|\nabla\varphi\|\le 1}\left\{\int_{\mathcal{X}}\varphi(x)d\mu(x)-\int_{\mathcal{X}}\varphi(x)d\nu(x)\right\}. \tag{4.50}$$

When $\mathcal{X}=\mathbb{R}$, the Wasserstein-1 distance has the closed-form expression

$$W_1(\mu,\nu)=\int_{-\infty}^{\infty}|F_\mu(x)-F_\nu(x)|dx \tag{4.51}$$

where F_μ and F_ν are the cumulative distribution functions of μ and ν, respectively. For $p=2$, the squared Wasserstein-2 distance is given by

$$W_2^2(\mu,\nu)=\inf_{\gamma\in\Pi(\mu,\nu)}\int_{\mathcal{X}\times\mathcal{X}} d(x,y)^2 d\gamma(x,y). \tag{4.52}$$

If μ and ν are Gaussian distributions $\mathcal{N}(m_1,\Sigma_1)$ and $\mathcal{N}(m_2,\Sigma_2)$, then

$$W_2^2(\mu,\nu)=\|m_1-m_2\|^2+\mathrm{Tr}(\Sigma_1+\Sigma_2-2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}). \tag{4.53}$$

The Wasserstein distance induces a metric on the space of probability measures, providing a notion of distance that is weaker than total variation but stronger than weak convergence. Specifically, if $\mu_n\to\mu$ in Wasserstein distance, then for any function f satisfying

$$|f(x)|\le C(1+d(x,x_0)^p), \tag{4.54}$$

it holds that

$$\int_{\mathcal{X}} f(x)d\mu_n(x) \to \int_{\mathcal{X}} f(x)d\mu(x). \tag{4.55}$$

The Wasserstein metric endows the space of probability measures with a Riemannian-like structure, where geodesics are given by **McCann interpolation**, defined as

$$\mu_t = ((1-t)\,\mathrm{id} + tT)_{\#}\mu_0, \quad t \in [0,1]. \tag{4.56}$$

For discrete distributions

$$\mu = \sum_{i=1}^{n} a_i \delta_{x_i}, \quad \nu = \sum_{j=1}^{m} b_j \delta_{y_j}, \tag{4.57}$$

the Wasserstein distance reduces to a linear programming problem

$$\min_{\gamma \in \Pi(\mu,\nu)} \sum_{i,j} \gamma_{ij} d(x_i, y_j)^p. \tag{4.58}$$

Computationally, Wasserstein distances are expensive to evaluate, but regularized approximations such as **Sinkhorn's algorithm** solve the entropically regularized problem

$$W_p^{\lambda}(\mu,\nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \sum_{i,j} \gamma_{ij} d(x_i, y_j)^p + \lambda \sum_{i,j} \gamma_{ij} \log \gamma_{ij}. \tag{4.59}$$

This approximation allows for efficient computation via iterative updates using the Sinkhorn-Knopp algorithm. The Wasserstein distance plays a crucial role in probability theory, statistics, machine learning, and functional analysis, providing a rigorous mathematical framework for comparing distributions with deep connections to convex geometry, measure theory, and differential geometry.

4.2 Spectral Regularization

The concept of spectral regularization, which refers to the preferential learning of low-frequency modes by neural networks before high-frequency modes, emerges from a combination of Fourier analysis, optimization theory, and the inherent properties of deep neural networks. This phenomenon is tightly connected to the functional approximation capabilities of neural networks and can be rigorously understood through the lens of Fourier decomposition and the gradient descent optimization process.

Literature Review: Jin et. al. (2025) [102] introduced a novel confusional spectral regularization technique to improve fairness in machine learning models. The study focuses on the spectral norm of the robust confusion matrix and proposes a method to control spectral properties, ensuring more robust and unbiased learning. It provides insights into how regularization can mitigate biases in classification tasks. Ye et. al. (2025) [103] applied spectral clustering with regularization to detect small clusters in complex networks. The work enhances spectral clustering techniques by integrating regularization methods, allowing improved performance in anomaly detection and community detection tasks. The approach significantly improves robustness in highly noisy data environments. Bhattacharjee and Bharadwaj (2025) [104] explored how spectral domain representations can benefit from autoencoder-based feature extraction combined with stochastic regularization techniques. The authors propose a Symmetric Autoencoder (SymAE) that enables better generalization of spectral features, particularly useful in high-dimensional data and deep learning applications. Wu et. al. (2025) [105] applied spectral regularization to geophysical data processing, specifically for high-resolution velocity spectrum analysis. The approach enhances the resolution of velocity estimation in seismic imaging by using hyperbolic Radon transform regularization, demonstrating how spectral regularization can benefit applications beyond traditional ML. Ortega et. al. (2025) [106]

applied Tikhonov regularization to atmospheric spectral analysis, optimizing gas retrieval strategies in high-resolution spectroscopic observations. The work significantly improves methane (CH4) and nitrous oxide (N2O) detection accuracy by reducing noise in spectral measurements, showcasing the impact of spectral regularization in remote sensing and environmental monitoring. Kazmi et. al. (2025) [107] proposed a spectral regularization-based federated learning model to improve robustness in cybersecurity threat detection. The model addresses the issue of non-IID data in SDN (Software Defined Networks) by utilizing spectral norm-based regularization within deep learning architectures. Zhao et. al. (2025) [108] introduced a regularized deep spectral clustering method, which enhances feature selection and clustering robustness. The authors utilize projected adaptive feature selection combined with spectral graph regularization, improving clustering accuracy and interpretability in high-dimensional datasets. Saranya and Menaka (2025) [109] integrated spectral regularization with quantum-based machine learning to analyze EEG signals for Autism Spectrum Disorder (ASD) detection. The proposed method improves spatial filtering and feature extraction using wavelet-based regularization, leading to more reliable EEG pattern recognition. Dhalbisoi et. al. (2024) [110] developed a Regularized Zero-Forcing (RZF) method for spectral efficiency optimization in beyond 5G networks. The authors demonstrate that spectral regularization techniques can significantly improve signal-to-noise ratios in wireless communication systems, optimizing data transmission in massive MIMO architectures. Wei et. al. (2025) [111] explored the use of spectral regularization in medical imaging, particularly in 3D near-infrared spectral tomography. The proposed model integrates regularized convolutional neural networks (CNNs) to improve tissue imaging resolution and accuracy, demonstrating an application of spectral regularization in biomedical engineering.

Let us define a target function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$, and its Fourier transform $\hat{f}(\boldsymbol{\xi})$ as

$$\hat{f}(\boldsymbol{\xi}) = \int_{\mathbb{R}^d} f(\mathbf{x}) e^{-i2\pi \boldsymbol{\xi}\cdot\mathbf{x}} \, d\mathbf{x} \tag{4.60}$$

This transform breaks down $f(\mathbf{x})$ into frequency components indexed by $\boldsymbol{\xi}$. In the context of deep learning, we seek to approximate $f(\mathbf{x})$ with a neural network output $f_{\text{NN}}(\mathbf{x};\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the set of trainable parameters. The loss function to be minimized is typically the mean squared error:

$$\mathcal{L}(\boldsymbol{\theta}) = \int_{\mathbb{R}^d} \left| f(\mathbf{x}) - f_{\text{NN}}(\mathbf{x};\boldsymbol{\theta}) \right|^2 \, d\mathbf{x} \tag{4.61}$$

We can equivalently express this loss in the Fourier domain, leveraging Parseval's theorem:

$$\mathcal{L}(\boldsymbol{\theta}) = \int_{\mathbb{R}^d} \left| \hat{f}(\boldsymbol{\xi}) - \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta}) \right|^2 \, d\boldsymbol{\xi} \tag{4.62}$$

To solve for $\boldsymbol{\theta}$, we employ gradient descent:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{4.63}$$

where η is the learning rate. The gradient of the loss function with respect to $\boldsymbol{\theta}$ is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = 2 \int_{\mathbb{R}^d} \left(\hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta}) - \hat{f}(\boldsymbol{\xi}) \right) \nabla_{\boldsymbol{\theta}} \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta}) \, d\boldsymbol{\xi} \tag{4.64}$$

At the core of this gradient descent process lies the behavior of the gradient $\nabla_{\boldsymbol{\theta}} \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta})$ with respect to the frequency components $\boldsymbol{\xi}$. For neural networks, particularly those with ReLU activations, the gradients of the output with respect to the parameters are expected to decay for high-frequency components. This can be approximated as

$$\mathcal{R}(\boldsymbol{\xi}) \sim \frac{1}{1 + \|\boldsymbol{\xi}\|^2} \tag{4.65}$$

which implies that the neural network is inherently more sensitive to low-frequency components of the target function during early iterations of training. This spectral decay is a direct consequence of the structure of the network's activations, which are more sensitive to low-frequency features due to their smoother, lower-order terms. To understand the role of the neural tangent kernel (NTK), which governs the linearized dynamics of the neural network, we define the NTK as

$$\Theta(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \sum_{i=1}^{P} \frac{\partial f_{\text{NN}}(\mathbf{x}; \boldsymbol{\theta})}{\partial \theta_i} \frac{\partial f_{\text{NN}}(\mathbf{x}'; \boldsymbol{\theta})}{\partial \theta_i} \tag{4.66}$$

The NTK essentially describes how the output of the network changes with respect to its parameters. The evolution of the network's output during training can be approximated by the solution to a linear system governed by the NTK. The output of the network at time t is given by

$$f_{\text{NN}}(\mathbf{x}; t) = \sum_k c_k \left(1 - e^{-\eta\lambda_k t}\right) \phi_k(\mathbf{x}) \tag{4.67}$$

where $\{\lambda_k\}$ are the eigenvalues of Θ, and $\{\phi_k(\mathbf{x})\}$ are the corresponding eigenfunctions. The eigenvalues λ_k determine the speed of convergence for each frequency mode, with low-frequency modes (large λ_k) converging more quickly than high-frequency ones (small λ_k):

$$1 - e^{-\eta\lambda_k t} \to 1 \quad \text{for large } \lambda_k \quad \text{and} \quad 1 - e^{-\eta\lambda_k t} \to 0 \quad \text{for small } \lambda_k \tag{4.68}$$

This differential learning rate for frequency components leads to the spectral regularization phenomenon, where the network learns the low-frequency components of the function first, and the high-frequency modes only begin to adapt once the low-frequency ones have been approximated with sufficient accuracy. In a more formal setting, the spectral bias can also be understood in terms of Sobolev spaces. A neural network function f_{NN} can be seen as a function in a Sobolev space $W^{m,2}$, where the norm of a function f in this space is defined as

$$\|f\|_{W^{m,2}}^2 = \int_{\mathbb{R}^d} \left(1 + \|\boldsymbol{\xi}\|^2\right)^m \left|\hat{f}(\boldsymbol{\xi})\right|^2 \, d\boldsymbol{\xi} \tag{4.69}$$

When training a neural network, the optimization process implicitly regularizes the higher-order Sobolev norms, meaning that the network will initially approximate the target function in terms of lower-order derivatives (which correspond to low-frequency modes). This can be expressed by introducing a regularization term in the loss function:

$$\mathcal{L}_{\text{eff}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda \|f_{\text{NN}}\|_{W^{m,2}}^2 \tag{4.70}$$

where λ is a regularization parameter that controls the trade-off between data fidelity and smoothness in the approximation.

Thus, spectral regularization emerges as a consequence of the network's architecture, the nature of gradient descent optimization, and the inherent smoothness of the functions that neural networks are capable of learning. The mathematical structure of the NTK and the regularization properties of the Sobolev spaces provide a rigorous framework for understanding why neural networks prioritize the learning of low-frequency modes, reinforcing the idea that neural networks are implicitly biased toward smooth, low-frequency approximations at the beginning of training. This insight has profound implications for the generalization behavior of neural networks and their capacity to approximate complex functions.

5 Game-theoretic formulations of Deep Neural Networks

Deep Neural Networks (DNNs) can be rigorously formulated within a game-theoretic framework by considering the interplay between competing objectives in optimization, adversarial robustness, and equilibrium strategies. The fundamental nature of DNNs involves a multi-agent optimization problem where various components interact within a high-dimensional function space. Formally, a deep neural network is parameterized by a set of weights $\theta \in \mathbb{R}^d$ and seeks to optimize a loss function $L : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}$, where the training data $(x, y) \sim P(x, y)$ is drawn from an unknown distribution. The optimization problem can be expressed as

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim P}[L(f_\theta(x), y)]. \tag{5.1}$$

A fundamental game-theoretic formulation of DNNs arises in adversarial training, where a deep network $f_\theta(x)$ is trained to minimize an objective function while an adversary seeks to maximize it. This setting leads to a two-player zero-sum game where the adversary introduces perturbations $\delta \in \mathcal{D}$ such that the resulting optimization problem is

$$\min_\theta \max_{\delta\in\mathcal{D}} \mathbb{E}_{(x,y)\sim P} \left[L(f_\theta(x + \delta), y)\right]. \tag{5.2}$$

The equilibrium of this adversarial game corresponds to a minimax solution, satisfying the condition

$$\theta^* = \arg\min_\theta \max_{\delta\in\mathcal{D}} L(f_\theta(x + \delta), y). \tag{5.3}$$

If the loss function satisfies convexity in θ and concavity in δ, then the minimax theorem guarantees the existence of an equilibrium solution:

$$\min_\theta \max_\delta L(\theta, \delta) = \max_\delta \min_\theta L(\theta, \delta). \tag{5.4}$$

A notable game-theoretic application of DNNs is in Generative Adversarial Networks (GANs), where two neural networks, a generator $G(z)$ and a discriminator $D(x)$, play a competitive game. The generator aims to produce realistic samples $G(z) \sim P_G$, while the discriminator seeks to distinguish between real and generated samples. The objective function for this adversarial game is

$$\min_G \max_D \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]. \tag{5.5}$$

The Nash equilibrium condition for this game is given by

$$D^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_G(x)}. \tag{5.6}$$

At equilibrium, the optimal generator satisfies

$$P_G(x) = P_{\text{data}}(x), \tag{5.7}$$

leading to a situation where the discriminator is unable to distinguish real from generated samples. If $D(x)$ is parameterized by ϕ, then training the discriminator corresponds to solving

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{x\sim P_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D_\phi(G_\theta(z)))]. \tag{5.8}$$

Given that neural networks can approximate arbitrary functions, their training can also be formulated in terms of multi-agent differential game theory. Consider a dynamic system where the evolution of parameters follows

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta, \phi), \tag{5.9}$$

$$\frac{d\phi}{dt} = \nabla_\phi \mathcal{L}(\theta, \phi). \tag{5.10}$$

The Hamiltonian for this system is given by

$$H(\theta, p) = \min_u \left[\nabla_\theta \mathcal{L} \cdot u + p \cdot f(\theta, u)\right], \tag{5.11}$$

where p is the costate variable and u represents control parameters. The Hamilton-Jacobi-Bellman equation characterizing optimal control in a reinforcement learning-based DNN training scenario is

$$\frac{\partial V}{\partial t} + \min_u \left[\nabla_\theta \mathcal{L} \cdot u + p \cdot f(\theta, u)\right] = 0. \tag{5.12}$$

Another approach to studying DNNs in a game-theoretic framework is through evolutionary game dynamics. Here, learning dynamics follow replicator equations given by

$$\frac{dx_i}{dt} = x_i \left(f_i - \bar{f}\right), \tag{5.13}$$

where x_i represents the probability of selecting strategy i, f_i is the fitness function for strategy i, and $\bar{f} = \sum_j x_j f_j$ is the average fitness. In the context of neural networks, the replicator equation governs weight updates as follows:

$$\frac{d\theta_i}{dt} = \theta_i \left(\nabla_{\theta_i} J - \frac{1}{n}\sum_j \nabla_{\theta_j} J\right). \tag{5.14}$$

This provides a mechanism for gradient-based optimization that incorporates selection and mutation principles. Another critical formulation involves Nash equilibria in non-cooperative learning settings, such as federated learning, where multiple learners optimize separate but interdependent loss functions. The Nash equilibrium conditions require that

$$\theta_i^* = \arg\min_{\theta_i} L_i(\theta_i, \theta_{-i}), \tag{5.15}$$

where θ_{-i} denotes the set of parameters for all players except i. A Nash equilibrium is attained when

$$\frac{\partial L_i}{\partial \theta_i} = 0, \quad \forall i. \tag{5.16}$$

Under convexity assumptions, Nash learning dynamics can be characterized by a gradient projection algorithm:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta^{(t)}). \tag{5.17}$$

Beyond Nash equilibria, deep neural networks can also be analyzed through variational inequalities, which generalize equilibrium concepts. Given a function $F : \mathbb{R}^d \to \mathbb{R}^d$, the variational inequality problem seeks θ^* such that

$$\langle F(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.18}$$

For adversarial training, the corresponding variational inequality formulation is

$$\langle \nabla_\theta L(\theta, \delta^*(\theta)), \theta - \theta^* \rangle \geq 0. \tag{5.19}$$

where $\delta^*(\theta)$ denotes the optimal adversarial perturbation satisfying

$$\delta^*(\theta) = \arg\max_{\delta \in \mathcal{D}} L(\theta, \delta). \tag{5.20}$$

This variational inequality provides a principled characterization of adversarial robustness in deep networks. The application of game-theoretic formulations to deep learning thus allows for a mathematically rigorous understanding of optimization, equilibrium conditions, and strategic interactions among network components.

5.1 Game-theoretic formulations of deep neural networks (DNNs) through Evolutionary Game Dynamics

Deep neural networks (DNNs) can be formulated within the framework of evolutionary game dynamics, where the optimization of network parameters is recast as a competitive evolutionary process in which different parameter configurations act as strategies that undergo selection, mutation, and adaptation over time. This perspective allows a rigorous characterization of learning dynamics through replicator equations, Nash equilibria, and mutation-selection balance, thereby linking deep learning to fundamental principles in game theory and statistical physics. Given a neural network parameterized by $\theta \in \Theta$, where Θ is the space of all possible weight configurations, the performance of the network is quantified by a fitness function $F(\theta)$, which is typically defined as the negative of the empirical loss function $L(\theta)$. This loss function can be expressed as an expectation over a given dataset D,

$$L(\theta) = \mathbb{E}_{(x,y)\sim D}[\ell(f_\theta(x), y)], \tag{5.21}$$

where ℓ represents the pointwise loss function, such as the squared error loss or cross-entropy loss. The fitness of a given parameter configuration θ is then defined as

$$F(\theta) = -L(\theta), \tag{5.22}$$

which ensures that lower loss corresponds to higher fitness. The evolutionary distribution of network parameters, denoted by $p(\theta, t)$, follows an evolutionary replicator-mutator equation,

$$\frac{dp(\theta, t)}{dt} = p(\theta, t)(F(\theta) - \bar{F}) + \sigma^2 \nabla^2 p(\theta, t), \tag{5.23}$$

where $\bar{F}$ is the mean fitness across all parameter configurations,

$$\bar{F} = \int_\Theta p(\theta, t) F(\theta) d\theta. \tag{5.24}$$

The first term in the equation represents selection dynamics, where parameter configurations with higher fitness proliferate, while the second term models mutations, accounting for stochastic noise in learning updates. The steady-state solution to this equation corresponds to an evolutionarily stable distribution $p^*(\theta)$, which satisfies

$$\mathbb{E}_{\theta' \sim p^*(\theta)}[F(\theta')] > \mathbb{E}_{\theta' \sim p(\theta)}[F(\theta')] \tag{5.25}$$

for any small perturbation $p(\theta)$ from $p^*(\theta)$. The learning process in deep networks can thus be viewed as an evolutionary competition in which weight distributions evolve toward stable equilibria or limit cycles. The standard stochastic gradient descent (SGD) update rule, given by

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta), \tag{5.26}$$

can be reinterpreted as a mean-field approximation of the replicator equation with stochastic diffusion,

$$\frac{d\theta}{dt} = -\nabla_\theta L(\theta) + \sigma\xi(t), \tag{5.27}$$

where $\xi(t)$ represents a stochastic perturbation modeling noise in the gradient updates. In the limit of infinitesimally small learning rate η, the SGD update can be seen as an Ornstein-Uhlenbeck process,

$$d\theta = -\nabla_\theta L(\theta)dt + \sigma dW_t, \tag{5.28}$$

where W_t denotes a Wiener process. The equilibrium distribution of network parameters is then governed by the Fokker-Planck equation,

$$\frac{\partial p(\theta,t)}{\partial t} = -\nabla_\theta \cdot (p(\theta,t)\nabla_\theta L(\theta)) + \frac{\sigma^2}{2}\nabla^2 p(\theta,t). \tag{5.29}$$

This equation describes the evolution of the probability density of weight configurations under a combination of gradient descent and stochastic exploration, which can be interpreted as a mutation-selection process in an evolutionary game. The steady-state solution $p^*(\theta)$ corresponds to a Boltzmann-Gibbs distribution,

$$p^*(\theta) = \frac{e^{-\beta L(\theta)}}{Z}, \tag{5.30}$$

where $\beta \propto 1/\sigma^2$ acts as an inverse temperature parameter, and Z is the partition function ensuring normalization. This establishes a direct connection between deep learning optimization and principles of statistical mechanics, whereby learning dynamics resemble an energy minimization process subject to thermal noise. In the case of generative adversarial networks (GANs), the evolutionary perspective is particularly insightful. A GAN consists of a generator G and a discriminator D, which play a two-player zero-sum game governed by the minimax objective,

$$\min_G \max_D \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]. \tag{5.31}$$

The training dynamics of GANs exhibit co-evolutionary behavior, which can be captured by the coupled replicator equations,

$$\frac{dp_G(\theta_G,t)}{dt} = p_G(\theta_G,t)(F_G(\theta_G) - \bar{F}_G) + \sigma^2\nabla^2 p_G(\theta_G,t), \tag{5.32}$$

$$\frac{dp_D(\theta_D,t)}{dt} = p_D(\theta_D,t)(F_D(\theta_D) - \bar{F}_D) + \sigma^2\nabla^2 p_D(\theta_D,t). \tag{5.33}$$

Unlike standard deep learning, GAN training often fails to converge to an equilibrium due to cycling behavior. This phenomenon can be analyzed through Hamiltonian game dynamics,

$$\frac{d\theta_G}{dt} = -\nabla_{\theta_G} L_G(\theta_G, \theta_D), \tag{5.34}$$

$$\frac{d\theta_D}{dt} = \nabla_{\theta_D} L_D(\theta_G, \theta_D). \tag{5.35}$$

These equations describe a conservative dynamical system, where learning trajectories follow closed orbits rather than converging to fixed points. This behavior is characteristic of zero-sum games and is mathematically analogous to the Red Queen effect in evolutionary biology, where competing species continually adapt without achieving a stable equilibrium.

5.2 Analysis of Deep neural networks (DNNs) through Variational Inequalities

Deep neural networks (DNNs) can be rigorously analyzed through the mathematical framework of variational inequalities, which provide a fundamental structure for understanding the equilibrium properties of optimization processes underlying deep learning. Variational inequalities (VIs) generalize optimization problems and offer a broader perspective on stability, convergence, and generalization in high-dimensional nonconvex settings. The problem of training a deep neural network can be formulated as a variational inequality by considering the minimization of a loss function $L : \mathbb{R}^d \to \mathbb{R}$, where the goal is to determine the optimal parameters θ^* satisfying the condition

$$\langle \nabla L(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.36}$$

This inequality characterizes the equilibrium conditions in the optimization landscape of deep learning. It expresses the fact that at the optimal point θ^*, any infinitesimal movement in the parameter space does not yield a lower value of the loss function. The gradient descent update rule, given by

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla L(\theta^{(k)}), \tag{5.37}$$

where $\eta > 0$ is the learning rate, can be rewritten as a **projection-based variational inequality** if constraints exist. Specifically, if the parameter space is constrained to a closed and convex set $\Theta \subset \mathbb{R}^d$, then the variational inequality formulation of the training process is

$$\langle \nabla L(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \Theta. \tag{5.38}$$

This variational inequality expresses the fact that θ^* is a stationary point of the constrained optimization problem, meaning that the projected gradient update

$$\theta^{(k+1)} = \text{Proj}_\Theta \left(\theta^{(k)} - \eta \nabla L(\theta^{(k)}) \right), \tag{5.39}$$

ensures that the sequence $\{\theta^{(k)}\}$ converges to the solution of the variational inequality under suitable assumptions on the loss function. A key property in variational inequalities is monotonicity. The operator associated with the variational inequality is given by

$$F(\theta) = \nabla L(\theta), \tag{5.40}$$

and monotonicity is defined by the condition

$$\langle F(\theta_1) - F(\theta_2), \theta_1 - \theta_2 \rangle \geq 0, \quad \forall \theta_1, \theta_2 \in \mathbb{R}^d. \tag{5.41}$$

For convex loss functions, $\nabla L(\theta)$ is a monotone operator, ensuring the uniqueness of the equilibrium point. If $\nabla L(\theta)$ is strongly monotone, satisfying

$$\langle \nabla L(\theta_1) - \nabla L(\theta_2), \theta_1 - \theta_2 \rangle \geq m \|\theta_1 - \theta_2\|^2, \quad \text{for some } m > 0, \tag{5.42}$$

then the optimization process exhibits **exponential convergence**, meaning that the iterates satisfy

$$\|\theta^{(k)} - \theta^*\| \leq C e^{-mk} \|\theta^{(0)} - \theta^*\|, \tag{5.43}$$

where C is a constant. This strong monotonicity property provides a rigorous foundation for understanding the rate of convergence of deep neural network training. In adversarial training, deep networks are often formulated as **saddle-point problems** of the form

$$\min_\theta \max_z \mathcal{L}(\theta, z), \tag{5.44}$$

where z represents an adversarial perturbation. The equilibrium conditions in this case are described by the system of variational inequalities

$$\langle \nabla_\theta \mathcal{L}(\theta^*, z^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta, \tag{5.45}$$

$$\langle \nabla_z \mathcal{L}(\theta^*, z^*), z - z^* \rangle \leq 0, \quad \forall z. \tag{5.46}$$

These inequalities characterize the stable equilibrium of the adversarial learning process, ensuring that neither the network nor the adversary can unilaterally improve their respective objectives. The analysis of stochastic gradient descent (SGD) in deep learning can also be conducted via variational inequalities. The expected gradient field in the presence of stochasticity is given by

$$F(\theta) = \mathbb{E}[\nabla L(\theta, \xi)], \tag{5.47}$$

where ξ represents a random variable encoding the data distribution. The variational inequality for SGD is then

$$\langle F(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.48}$$

Stochastic approximation methods ensure convergence to the equilibrium point under conditions on the variance of the gradient estimator. Specifically, if $F(\theta)$ is **co-coercive**, meaning there exists $L > 0$ such that

$$\|F(\theta_1) - F(\theta_2)\|^2 \leq L \langle F(\theta_1) - F(\theta_2), \theta_1 - \theta_2 \rangle, \tag{5.49}$$

then projected SGD satisfies

$$\mathbb{E}[\|\theta^{(k)} - \theta^*\|^2] \leq \frac{L}{2m} \frac{\sigma^2}{k}, \tag{5.50}$$

where σ^2 is the variance of the stochastic gradient noise. This establishes a rigorous bound on the convergence rate of SGD in terms of variational inequalities. The generalization properties of deep networks can also be rigorously studied using variational inequalities. Given a distribution shift perturbation δ, the parameter shift $\theta' = \theta + \delta$ satisfies the **generalization bound**

$$\mathbb{E}[L(\theta')] - \mathbb{E}[L(\theta)] \leq C \|\delta\|, \tag{5.51}$$

where the constant C is determined by the Lipschitz continuity of the gradient field. This bound ensures that the network's performance remains stable under small perturbations in the data distribution. The variational inequality framework provides a **unified perspective** on optimization, equilibrium, and generalization in deep learning, offering rigorous theoretical insights into the dynamics of training and stability of neural networks.

5.3 Optimal control in reinforcement learning (RL)-based deep neural network (DNN) training

Let $\theta_t \in \mathbb{R}^n$ represent the weight parameters of a deep neural network (DNN) at time step t. The evolution of these parameters over discrete time is governed by a controlled dynamical system, where the control input $u_t \in \mathbb{R}^n$ represents the weight update at each iteration. The fundamental objective of reinforcement learning (RL)-based DNN training is to determine an optimal control policy u_t^* that minimizes a given performance criterion over a finite or infinite training horizon. The system dynamics are given by the equation

$$\theta_{t+1} = \theta_t + u_t. \tag{5.52}$$

Let $J(\theta_0, u)$ denote the cumulative loss function associated with a given control sequence $\{u_t\}_{t=0}^T$, defined as

$$J(\theta_0, u) = \sum_{t=0}^{T} c(\theta_t, u_t), \tag{5.53}$$

where $c(\theta_t, u_t)$ is the instantaneous cost function. In the context of neural network training, the cost function is typically the empirical risk over a dataset $\mathcal{D}$, given by

$$c(\theta_t, u_t) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\ell(f_{\theta_t}(x), y)\right], \tag{5.54}$$

where ℓ is a loss function such as cross-entropy or mean squared error, and $f_{\theta_t}(x)$ represents the neural network's output for an input x. The optimal control problem consists of finding a control sequence $\{u_t^*\}_{t=0}^T$ such that

$$J^*(\theta_0) = \min_u J(\theta_0, u). \tag{5.55}$$

Applying the principle of dynamic programming, the optimal cost-to-go function is defined recursively as

$$V(\theta_t) = \min_{u_t} \left\{c(\theta_t, u_t) + V(\theta_{t+1})\right\}. \tag{5.56}$$

Expanding $\theta_{t+1} = \theta_t + u_t$, the Bellman equation takes the form

$$V(\theta_t) = \min_{u_t} \left\{c(\theta_t, u_t) + V(\theta_t + u_t)\right\}. \tag{5.57}$$

Considering the continuous-time limit where the weight updates are infinitesimal, i.e., $u_t \to \dot{\theta}(t)dt$, we obtain the Hamilton-Jacobi-Bellman (HJB) equation

$$\frac{\partial V}{\partial t} + \min_u \left[\frac{\partial V}{\partial \theta} u + c(\theta, u)\right] = 0. \tag{5.58}$$

The optimal control $u^*(t)$ satisfies

$$u^*(t) = \arg\min_u \left[\frac{\partial V}{\partial \theta} u + c(\theta, u)\right]. \tag{5.59}$$

Alternatively, the problem can be analyzed using Pontryagin's Maximum Principle. Define the Hamiltonian

$$H(\theta, u, \lambda) = c(\theta, u) + \lambda^\top (\theta + u), \tag{5.60}$$

where λ is the costate variable that evolves according to the adjoint equation

$$\frac{d\lambda}{dt} = -\frac{\partial H}{\partial \theta}. \tag{5.61}$$

For optimality, the control must satisfy

$$\frac{\partial H}{\partial u} = \frac{\partial c}{\partial u} + \lambda = 0. \tag{5.62}$$

Solving for u^*, we obtain

$$u^* = -\left(\frac{\partial^2 c}{\partial u^2}\right)^{-1} \left(\frac{\partial c}{\partial u} + \lambda\right). \tag{5.63}$$

In the reinforcement learning framework, the optimal control formulation is connected to policy optimization, where the policy $\pi(a_t|s_t)$ determines the action a_t given the state s_t. The expected return under policy π is given by

$$J(\pi) = \mathbb{E}_{\tau\sim\pi} \left[\sum_{t=0}^T \gamma^t R_t\right], \tag{5.64}$$

where R_t is the reward function and $\gamma \in (0, 1]$ is the discount factor. The policy gradient theorem states that

$$\nabla_\theta J(\pi) = \mathbb{E}_{\tau\sim\pi} \left[\sum_{t=0}^T \nabla_\theta \log \pi(a_t|s_t) Q^\pi(s_t, a_t)\right], \tag{5.65}$$

where $Q^{\pi}(s_t, a_t)$ is the state-action value function satisfying the Bellman equation

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[R_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t, a_t\right]. \tag{5.66}$$

By applying the HJB equation, the optimal policy update rule is derived as

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} J(\pi), \tag{5.67}$$

where η is the learning rate, often adaptively adjusted based on second-order curvature information such as in natural gradient descent. More formally, using the Fisher information matrix $F(\theta)$, the optimal update rule can be written as

$$\theta_{t+1} = \theta_t - \eta F^{-1}(\theta) \nabla_{\theta} J(\pi). \tag{5.68}$$

This establishes the link between optimal control and policy optimization in reinforcement learning, providing a rigorous theoretical foundation for RL-based DNN training.

6 Optimal transport theory in Deep Neural Networks

Optimal transport theory provides a mathematically rigorous framework for measuring distances between probability distributions by solving a constrained optimization problem that minimizes the cost of transporting mass from one distribution to another. Consider two probability measures μ and ν defined on a common measurable space $(\Omega, \mathcal{F})$, where μ has support $\mathcal{X} \subseteq \mathbb{R}^d$ and ν has support $\mathcal{Y} \subseteq \mathbb{R}^d$. Given a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$, the optimal transport problem seeks to determine a transport plan that minimizes the total cost of transforming μ into ν. In the Monge formulation, this problem is expressed as

$$\inf_{T:\mathcal{X}\to\mathcal{Y}} \int_{\mathcal{X}} c(x, T(x))d\mu(x), \tag{6.1}$$

subject to the constraint that the transport map T satisfies the push-forward condition

$$T_{\#}\mu = \nu, \tag{6.2}$$

which ensures that the measure ν is obtained by pushing forward μ through T, i.e., for any measurable set $A \subseteq \mathcal{Y}$,

$$\nu(A) = \mu(T^{-1}(A)). \tag{6.3}$$

The Monge problem is often ill-posed since a transport map may not exist. To alleviate this issue, the Kantorovich relaxation introduces a coupling γ between μ and ν, where γ is a probability measure on $\mathcal{X} \times \mathcal{Y}$ satisfying the marginal constraints

$$\int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x), \quad \int_{\mathcal{X}} d\gamma(x, y) = d\nu(y). \tag{6.4}$$

The Kantorovich formulation of the optimal transport problem is thus

$$\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y)d\gamma(x, y), \tag{6.5}$$

where $\Pi(\mu, \nu)$ is the set of all couplings satisfying the marginal constraints. A commonly used cost function is the p-th power of the Euclidean distance, i.e., $c(x, y) = \|x - y\|^p$, which leads to the Wasserstein distance of order p, given by

$$W_p(\mu, \nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x - y\|^p d\gamma(x, y) \right)^{\frac{1}{p}}. \tag{6.6}$$

The Wasserstein distance defines a proper metric on the space of probability measures, provided that μ and ν have finite p-th moments, i.e.,

$$\int_{\mathcal{X}} \|x\|^p d\mu(x) < \infty, \quad \int_{\mathcal{Y}} \|y\|^p d\nu(y) < \infty. \tag{6.7}$$

In deep neural networks, optimal transport plays a crucial role in generative modeling, where it provides a more stable alternative to divergence-based training objectives. In Wasserstein generative adversarial networks (WGANs), the standard Jensen-Shannon divergence is replaced by the Wasserstein-1 distance, leading to the objective

$$\sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{z \sim P_z}[f(G(z))], \tag{6.8}$$

where f is a Lipschitz-1 function, P_r is the real data distribution, P_z is the latent distribution, and $G : Z \to X$ is the generator function. The use of the Wasserstein distance improves gradient behavior and mitigates mode collapse, which is a common issue in standard GAN training. In domain adaptation, optimal transport is used to align source and target distributions by finding a transport plan γ that minimizes the Wasserstein distance between them. Given a source distribution P_s and a target distribution P_t, the goal is to find a transport map T such that

$$T_{\#}P_s \approx P_t, \tag{6.9}$$

which is often achieved by solving

$$\inf_{\gamma \in \Pi(P_s, P_t)} \int_{\mathcal{X} \times \mathcal{Y}} \|x - y\|^p d\gamma(x, y). \tag{6.10}$$

To ensure computational efficiency, entropy-regularized optimal transport introduces a regularization term based on the Kullback-Leibler divergence, leading to the Sinkhorn distance

$$W_\lambda(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) d\gamma(x, y) + \lambda KL(\gamma \| \mu \otimes \nu). \tag{6.11}$$

The regularization parameter $\lambda > 0$ controls the trade-off between accuracy and computational efficiency. The Sinkhorn algorithm, which iteratively updates the dual potentials using Bregman projections, provides an efficient means of computing approximate solutions to the OT problem. Optimal transport has also been applied to learning energy-based models, where it is used to define a structured loss function for training probability models. In probabilistic autoencoders, OT distances provide a principled approach for matching latent and data distributions. In Bayesian deep learning, OT-based priors ensure that posterior distributions maintain smooth and stable representations, improving generalization in uncertainty estimation tasks. The OT-based barycenter problem arises in multi-modal learning, where the goal is to compute a central distribution that minimizes the sum of Wasserstein distances to a set of given measures $\mu_1, \mu_2, \ldots, \mu_n$. The barycenter ν is defined as

$$\arg\min_{\nu} \sum_{i=1}^{n} W_p(\nu, \mu_i)^p, \tag{6.12}$$

which generalizes the notion of averaging to probability distributions. Applications of OT barycenters include meta-learning, ensemble methods, and federated learning, where a global model must aggregate information from multiple local distributions while minimizing divergence. The theoretical properties of optimal transport, including dual formulations and regularity results, provide strong guarantees for its application in deep neural networks. The dual formulation of the Wasserstein distance is given by the Kantorovich-Rubinstein theorem, which states that for W_1,

$$W_1(\mu, \nu) = \sup_{\|f\|_L \leq 1} \int_{\mathcal{X}} f(x) d\mu(x) - \int_{\mathcal{Y}} f(y) d\nu(y). \tag{6.13}$$

This dual representation allows gradient-based optimization methods to efficiently approximate OT distances in neural network training. Furthermore, recent advances in computational optimal transport, including sliced Wasserstein distances and entropic maps, have enabled scalable implementations suitable for high-dimensional deep learning applications.

Optimal transport thus provides a fundamental mathematical framework for distributional matching in deep learning. By leveraging Wasserstein distances and entropy-regularized formulations, OT enables more stable training, improved generalization, and better interpretability in neural networks, making it an essential tool for modern deep learning methodologies.

6.1 Jensen-Shannon divergence

The Jensen-Shannon divergence (JSD) is a rigorously defined, symmetric, and bounded measure of dissimilarity between two probability distributions P and Q. It is constructed from the Kullback-Leibler divergence (KLD) but incorporates a mixture distribution to ensure symmetry and finiteness. The JSD is defined as:

$$\mathrm{JSD}(P, Q) = \frac{1}{2} D_{KL}(P \| M) + \frac{1}{2} D_{KL}(Q \| M), \tag{6.14}$$

where M is the average distribution of P and Q, given by:

$$M = \frac{1}{2}(P + Q). \tag{6.15}$$

Here, $D_{KL}(P\|M)$ and $D_{KL}(Q\|M)$ are the Kullback-Leibler divergences between P and M, and Q and M, respectively. The KLD for two discrete probability distributions P and M is defined as:

$$D_{KL}(P \| M) = \sum_i P(i) \log\left(\frac{P(i)}{M(i)}\right), \tag{6.16}$$

where $P(i)$ and $M(i)$ are the probabilities assigned to the i-th event by P and M, respectively. For continuous probability distributions, the KLD is expressed as:

$$D_{KL}(P \| M) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{m(x)}\right) dx, \tag{6.17}$$

where $p(x)$ and $m(x)$ are the probability density functions of P and M, respectively. Substituting the definition of M into the JSD, we obtain:

$$\mathrm{JSD}(P, Q) = \frac{1}{2} \sum_i P(i) \log\left(\frac{P(i)}{\frac{1}{2}(P(i) + Q(i))}\right) + \frac{1}{2} \sum_i Q(i) \log\left(\frac{Q(i)}{\frac{1}{2}(P(i) + Q(i))}\right). \tag{6.18}$$

For continuous distributions, this becomes:

$$\mathrm{JSD}(P, Q) = \frac{1}{2} \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{\frac{1}{2}(p(x) + q(x))}\right) dx + \frac{1}{2} \int_{-\infty}^{\infty} q(x) \log\left(\frac{q(x)}{\frac{1}{2}(p(x) + q(x))}\right) dx. \tag{6.19}$$

The JSD can also be expressed in terms of Shannon entropy. Let $H(P)$ denote the Shannon entropy of P, defined for discrete distributions as:

$$H(P) = -\sum_i P(i) \log P(i). \tag{6.20}$$

For continuous distributions, the differential entropy is:

$$H(P) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx. \tag{6.21}$$

Using entropy, the JSD can be rewritten as:

$$\mathrm{JSD}(P, Q) = H\left(\frac{P+Q}{2}\right) - \frac{1}{2}H(P) - \frac{1}{2}H(Q). \tag{6.22}$$

This formulation demonstrates that the JSD measures the difference between the entropy of the average distribution M and the average of the entropies of P and Q. The JSD is bounded between 0 and 1 when the logarithm base is 2, as:

$$0 \leq \mathrm{JSD}(P, Q) \leq 1. \tag{6.23}$$

The lower bound is achieved when $P = Q$, and the upper bound is approached when P and Q are maximally different. The square root of the JSD is a metric, satisfying the triangle inequality:

$$\sqrt{\mathrm{JSD}(P, Q)} \leq \sqrt{\mathrm{JSD}(P, R)} + \sqrt{\mathrm{JSD}(R, Q)}, \tag{6.24}$$

for any probability distribution R. This property makes the JSD particularly useful in applications requiring a distance metric, such as clustering and classification. The JSD is also related to the total variation distance $\delta(P, Q)$ by the inequality:

$$\delta(P, Q) \leq \sqrt{2 \cdot \mathrm{JSD}(P, Q)}. \tag{6.25}$$

This relationship connects the JSD to other statistical distances, providing a bridge between information-theoretic and measure-theoretic perspectives. The JSD is widely used in machine learning, bioinformatics, and statistics due to its symmetry, boundedness, and interpretability. It is particularly useful for comparing empirical distributions, as it avoids the infinite values that can arise with the KLD. The JSD can also be generalized to compare more than two distributions. For n distributions $P_1, P_2, \ldots, P_n$, the generalized JSD is:

$$\mathrm{JSD}(P_1, P_2, \ldots, P_n) = H\left(\frac{1}{n}\sum_{i=1}^{n} P_i\right) - \frac{1}{n}\sum_{i=1}^{n} H(P_i). \tag{6.26}$$

This generalization retains the properties of the standard JSD, including symmetry and boundedness. The JSD is also closely related to the concept of mutual information, as it can be interpreted as the mutual information between a random variable and a mixture distribution. Specifically, for two distributions P and Q, the JSD can be expressed as:

$$\mathrm{JSD}(P, Q) = I(X; Y), \tag{6.27}$$

where X is a random variable with distribution P or Q, and Y is a binary indicator variable selecting between P and Q. This interpretation provides a deeper understanding of the JSD as a measure of information overlap. In summary, the Jensen-Shannon divergence is a versatile and mathematically rigorous tool for quantifying the difference between probability distributions, with applications across numerous scientific and engineering disciplines. Its properties make it a preferred choice in many scenarios where symmetric and bounded divergence measures are required.

6.2 Matching Latent and Data Distributions in Probabilistic Autoencoders

Optimal Transport (OT) distances provide a mathematically rigorous framework for comparing probability distributions, making them particularly suitable for aligning latent and data distributions in Probabilistic Autoencoders (PAEs). The core idea of OT is to find the most efficient way to transform one distribution into another, where efficiency is quantified by a cost function. In the context of PAEs, this translates to minimizing the discrepancy between the encoded latent distribution $q_\phi(z|x)$ and the prior distribution $p(z)$, as well as between the decoded data distribution $p_\theta(x|z)$ and the true data distribution $p_{\text{data}}(x)$.

Let μ and ν be two probability measures defined on metric spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively. The OT problem seeks a coupling $\pi \in \Pi(\mu, \nu)$ that minimizes the total transportation cost:

$$W_c(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, d\pi(x, y), \tag{6.28}$$

where $c(x, y)$ is a cost function, typically the Euclidean distance $c(x, y) = \|x - y\|^p$ for $p \geq 1$. The p-Wasserstein distance is a special case of OT, defined as:

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x - y\|^p\, d\pi(x, y) \right)^{1/p}. \tag{6.29}$$

In PAEs, the latent distribution $q_\phi(z|x)$ is often modeled as a Gaussian $\mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$, and the prior $p(z)$ is typically a standard Gaussian $\mathcal{N}(0, I)$. The OT distance between these distributions can be computed using the 2-Wasserstein distance, which has a closed-form expression for Gaussian distributions:

$$W_2^2(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)) = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}). \tag{6.30}$$

For the decoded data distribution $p_\theta(x|z)$, the OT distance to the true data distribution $p_{\text{data}}(x)$ can be approximated using discrete OT methods. Given samples $\{x_i\}_{i=1}^N$ from $p_{\text{data}}(x)$ and $\{\hat{x}_i\}_{i=1}^N$ from $p_\theta(x|z)$, the empirical OT problem becomes:

$$W_c(\hat{\mu}, \nu) = \min_{\pi \in \Pi(\hat{\mu},\nu)} \sum_{i,j} c(\hat{x}_i, x_j)\pi_{ij}, \tag{6.31}$$

where $\hat{\mu} = \frac{1}{N}\sum_{i=1}^N \delta_{\hat{x}_i}$ and $\nu = \frac{1}{N}\sum_{i=1}^N \delta_{x_i}$ are empirical measures. The coupling π is a doubly stochastic matrix satisfying $\sum_i \pi_{ij} = \frac{1}{N}$ and $\sum_j \pi_{ij} = \frac{1}{N}$. The OT distance can be incorporated into the PAE objective function as a regularization term. The overall loss function $\mathcal{L}$ combines the reconstruction loss and the OT-based regularization:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{x\sim p_{\text{data}}}[-\log p_\theta(x|z)] + \lambda W_2^2(q_\phi(z|x), p(z)) + \gamma W_c(p_\theta(x|z), p_{\text{data}}(x)), \tag{6.32}$$

where λ and γ are hyperparameters controlling the strength of the regularization terms. The first term ensures accurate reconstruction, while the second and third terms enforce consistency between the latent and prior distributions, and between the decoded and true data distributions, respectively. The OT framework also allows for the use of entropy-regularized OT, which introduces a regularization term to the OT problem to make it computationally tractable. The entropy-regularized OT problem is given by:

$$W_{c,\epsilon}(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \left(\int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, d\pi(x, y) + \epsilon H(\pi) \right) \tag{6.33}$$

where $H(\pi) = \int_{\mathcal{X}\times\mathcal{Y}} \pi(x, y) \log \pi(x, y)\, dx\, dy$ is the entropy of the coupling π, and $\epsilon > 0$ is the regularization parameter. This formulation leads to the Sinkhorn algorithm, which provides an efficient way to compute approximate OT distances.

In summary, OT distances offer a principled and mathematically rigorous approach for matching latent and data distributions in PAEs. By minimizing the Wasserstein distance between distributions, PAEs can achieve better alignment between the encoded latent space and the prior, as well as between the decoded data and the true data distribution. This results in more robust and interpretable models, with improved generative and reconstructive capabilities.

6.2.1 Probabilistic Autoencoders

Probabilistic autoencoders are generative models that combine the principles of autoencoders with probabilistic frameworks to learn a latent representation of data. Let $\mathbf{x} \in \mathbb{R}^D$ be the observed data, and $\mathbf{z} \in \mathbb{R}^d$ be the latent variable, where $d \ll D$. The goal is to model the joint distribution $p(\mathbf{x}, \mathbf{z})$, which can be factorized as:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}). \tag{6.34}$$

Here, $p(\mathbf{z})$ is the prior distribution over the latent space, often chosen as a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and $p(\mathbf{x}|\mathbf{z})$ is the likelihood function, which models the data generation process. The encoder in a probabilistic autoencoder approximates the posterior distribution $p(\mathbf{z}|\mathbf{x})$, which is typically intractable. Instead, a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is introduced, parameterized by ϕ, to approximate the true posterior. This distribution is often chosen as a Gaussian $\mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$, where $\mu_\phi(\mathbf{x})$ and $\Sigma_\phi(\mathbf{x})$ are outputs of a neural network. The decoder, parameterized by θ, models the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$, which is also often Gaussian for continuous data:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})). \tag{6.35}$$

The learning objective is to maximize the marginal likelihood $p(\mathbf{x})$, which can be expressed as:

$$p(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z}. \tag{6.36}$$

However, this integral is intractable due to the high-dimensional latent space. Instead, the evidence lower bound (ELBO) is maximized, which is derived using variational inference:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \mathrm{ELBO}. \tag{6.37}$$

The first term in the ELBO is the reconstruction loss, which encourages the model to accurately reconstruct the input data. The second term is the Kullback-Leibler (KL) divergence between the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$, which regularizes the latent space to match the prior distribution. The ELBO can be rewritten as:

$$\mathrm{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \frac{1}{2}\sum_{i=1}^{d}\left(\log \sigma_{\phi,i}^2(\mathbf{x}) - \sigma_{\phi,i}^2(\mathbf{x}) - \mu_{\phi,i}^2(\mathbf{x}) + 1\right), \tag{6.38}$$

where $\mu_{\phi,i}(\mathbf{x})$ and $\sigma_{\phi,i}^2(\mathbf{x})$ are the mean and variance of the i-th dimension of the latent variable $\mathbf{z}$. The parameters ϕ and θ are optimized using stochastic gradient descent. The expectation term in the ELBO is approximated using Monte Carlo sampling. For a single sample $\mathbf{z}^{(l)} \sim q_\phi(\mathbf{z}|\mathbf{x})$, the ELBO can be approximated as:

$$\mathrm{ELBO} \approx \log p_\theta(\mathbf{x}|\mathbf{z}^{(l)}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})). \tag{6.39}$$

To enable backpropagation through the sampling process, the reparameterization trick is used. Specifically, $\mathbf{z}$ is reparameterized as:

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \epsilon \odot \sigma_\phi(\mathbf{x}), \tag{6.40}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ denotes element-wise multiplication. This allows gradients to flow through the sampling process, enabling efficient optimization. The reconstruction loss $\log p_\theta(\mathbf{x}|\mathbf{z})$ depends on the type of data. For continuous data, it is often the negative log-likelihood of a Gaussian distribution:

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = -\frac{1}{2}\|\mathbf{x} - \mu_\theta(\mathbf{z})\|_2^2 + \text{constant}. \tag{6.41}$$

For binary data, the Bernoulli distribution is used, and the reconstruction loss becomes:

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{D} x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i), \tag{6.42}$$

where $\hat{x}_i$ is the i-th element of the reconstructed data $\mu_\theta(\mathbf{z})$. The KL divergence term $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ can be computed analytically for Gaussian distributions. For a standard Gaussian prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and a Gaussian variational posterior $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$, the KL divergence is:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\phi(\mathbf{x})) + \|\mu_\phi(\mathbf{x})\|_2^2 - d - \log\det(\Sigma_\phi(\mathbf{x}))\right). \tag{6.43}$$

If the variational posterior is diagonal, i.e., $\Sigma_\phi(\mathbf{x}) = \mathrm{diag}(\sigma_{\phi,1}^2(\mathbf{x}), \ldots, \sigma_{\phi,d}^2(\mathbf{x}))$, the KL divergence simplifies to:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \frac{1}{2}\sum_{i=1}^{d}\left(\sigma_{\phi,i}^2(\mathbf{x}) + \mu_{\phi,i}^2(\mathbf{x}) - 1 - \log\sigma_{\phi,i}^2(\mathbf{x})\right). \tag{6.44}$$

The optimization of the ELBO is performed using gradient-based methods. The gradients of the ELBO with respect to ϕ and θ are computed using automatic differentiation. For a mini-batch of data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\}$, the stochastic gradient estimate of the ELBO is:

$$\nabla_{\phi,\theta}\mathrm{ELBO} \approx \frac{1}{B}\sum_{b=1}^{B}\nabla_{\phi,\theta}\left(\log p_\theta(\mathbf{x}^{(b)}|\mathbf{z}^{(b)}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(b)})\|p(\mathbf{z}))\right), \tag{6.45}$$

where $\mathbf{z}^{(b)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(b)})$. The probabilistic autoencoder can be extended to more complex architectures, such as hierarchical models with multiple layers of latent variables. In such cases, the latent variables are structured hierarchically, and the joint distribution is factorized as:

$$p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_L) = p(\mathbf{x}|\mathbf{z}_1)\prod_{l=1}^{L-1} p(\mathbf{z}_l|\mathbf{z}_{l+1})p(\mathbf{z}_L), \tag{6.46}$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_L$ are the latent variables at different levels of the hierarchy. The variational posterior is also factorized hierarchically:

$$q_\phi(\mathbf{z}_1, \ldots, \mathbf{z}_L|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x})\prod_{l=2}^{L} q_\phi(\mathbf{z}_l|\mathbf{z}_{l-1}). \tag{6.47}$$

The ELBO for hierarchical models includes additional KL divergence terms for each level of the hierarchy:

$$\mathrm{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}_1,\ldots,\mathbf{z}_L|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}_1)\right] - \sum_{l=1}^{L} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_l|\mathbf{z}_{l-1})\|p(\mathbf{z}_l|\mathbf{z}_{l+1})). \tag{6.48}$$

In summary, probabilistic autoencoders provide a principled framework for learning latent representations of data by combining the strengths of autoencoders and probabilistic models. The ELBO serves as the optimization objective, balancing reconstruction accuracy and regularization of the latent space. The reparameterization trick enables efficient gradient-based optimization, and the framework can be extended to hierarchical models for more complex data structures.

6.2.2 2-Wasserstein distance

The 2-Wasserstein distance, also known as the Earth Mover's Distance (EMD) in the case of $p = 2$, is a metric used to quantify the difference between two probability distributions μ and ν defined on a metric space (X, d). It is a specific instance of the more general p-Wasserstein distance, which is defined for $p \geq 1$. The 2-Wasserstein distance is particularly significant due to its geometric interpretation and its applications in optimal transport theory, statistics, and machine learning.

Given two probability measures μ and ν on X, the 2-Wasserstein distance $W_2(\mu, \nu)$ is defined as the infimum of the expected squared distance between random variables X and Y, where X is distributed according to μ and Y is distributed according to ν, over all possible joint distributions π of (X, Y) with marginals μ and ν. Mathematically, this is expressed as:

$$W_2(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{X \times X} d(x, y)^2 \, d\pi(x, y) \right)^{1/2}, \tag{6.49}$$

where $\Pi(\mu, \nu)$ denotes the set of all joint probability measures π on $X \times X$ with marginals μ and ν, i.e., for any measurable sets $A, B \subset X$, $\pi(A \times X) = \mu(A)$ and $\pi(X \times B) = \nu(B)$. The distance $d(x, y)$ is the metric on the space X, and the integral $\int_{X \times X} d(x, y)^2 \, d\pi(x, y)$ represents the expected squared distance under the coupling π. The 2-Wasserstein distance can also be expressed in terms of the cumulative distribution functions (CDFs) of μ and ν when $X = \mathbb{R}$. Let F_μ and F_ν be the CDFs of μ and ν, respectively. Then, the 2-Wasserstein distance is given by:

$$W_2(\mu, \nu) = \left(\int_0^1 |F_\mu^{-1}(q) - F_\nu^{-1}(q)|^2 \, dq \right)^{1/2}, \tag{6.50}$$

where F_μ^{-1} and F_ν^{-1} are the quantile functions (inverse CDFs) of μ and ν, respectively. This formulation is particularly useful in one-dimensional settings, as it reduces the problem to computing the integral of the squared difference between the quantile functions. In the case where μ and ν are absolutely continuous with respect to the Lebesgue measure, with probability density functions f_μ and f_ν, the 2-Wasserstein distance can be related to the optimal transport map T that pushes μ forward to ν, i.e., $T_\# \mu = \nu$. The map T minimizes the transport cost:

$$W_2(\mu, \nu) = \left(\int_X |x - T(x)|^2 \, d\mu(x) \right)^{1/2}. \tag{6.51}$$

The optimal transport map T is often characterized by the Monge-Ampère equation, which relates the densities f_μ and f_ν through the Jacobian determinant of T:

$$f_\mu(x) = f_\nu(T(x)) \cdot |\det(\nabla T(x))|. \tag{6.52}$$

In higher dimensions, the 2-Wasserstein distance is more challenging to compute, but it can be approximated using numerical methods such as linear programming, entropic regularization, or Sinkhorn iterations. The dual formulation of the 2-Wasserstein distance, derived from Kantorovich duality, provides another perspective:

$$W_2(\mu, \nu)^2 = \sup_{\phi, \psi} \left(\int_X \phi(x) \, d\mu(x) + \int_X \psi(y) \, d\nu(y) \right), \tag{6.53}$$

where the supremum is taken over all pairs of continuous functions ϕ and ψ on X satisfying the constraint:

$$\phi(x) + \psi(y) \leq d(x, y)^2 \quad \forall x, y \in X. \tag{6.54}$$

This dual formulation is particularly useful in theoretical analyses and provides a connection to convex optimization. The 2-Wasserstein distance has several important properties, including symmetry, non-negativity, and the triangle inequality, making it a true metric on the space of probability measures with finite second moments. It is also sensitive to the geometry of the underlying space X, as it incorporates the metric $d(x, y)$ directly into its definition. This geometric sensitivity makes it particularly useful in applications such as image processing, where the spatial arrangement of pixels is crucial.

In summary, the 2-Wasserstein distance is a powerful tool for comparing probability distributions, with deep connections to optimal transport theory, convex analysis, and geometry. Its rigorous mathematical formulation and rich theoretical properties make it a cornerstone of modern probability and statistics.

6.3 Optimal Transport (OT)-based priors in Bayesian Deep Learning

The incorporation of Optimal Transport (OT)-based priors into Bayesian Deep Learning frameworks ensures that posterior distributions maintain smooth and stable representations by leveraging the rigorous mathematical properties of the Wasserstein distance and its associated geometric structure. This approach is grounded in measure theory, functional analysis, and convex optimization, providing a robust foundation for uncertainty estimation tasks. Below, we provide a deeper and more mathematically rigorous exposition.

Let (X, d) be a complete separable metric space, and let $P_p(X)$ denote the space of probability measures on X with finite p-th moments. For $P, Q \in P_p(X)$, the p-Wasserstein distance $W_p(P, Q)$ is defined as:

$$W_p(P,Q) = \left(\inf_{\pi \in \Pi(P,Q)} \int_{X \times X} d(x,y)^p \, d\pi(x,y) \right)^{1/p} \tag{6.55}$$

where $\Pi(P, Q)$ is the set of all couplings (joint distributions) with marginals P and Q. The Wasserstein distance induces a metric on $P_p(X)$, endowing it with a geometric structure that is sensitive to the underlying topology of X. This sensitivity ensures that the Wasserstein distance captures not only global but also local differences between distributions, making it particularly suitable for regularizing posterior distributions in Bayesian inference. In Bayesian Deep Learning, the prior $p(\theta)$ and the posterior $q(\theta)$ are probability measures over the parameter space Θ. To ensure smoothness and stability of the posterior, we introduce an OT-based regularization term into the variational inference objective. Specifically, we augment the evidence lower bound (ELBO) with a Wasserstein penalty:

$$L_{\text{ELBO}}(q) = \mathbb{E}_q(\theta) \left[\log p(D \mid \theta) \right] - \text{KL}(q(\theta) \parallel p(\theta)) + \lambda W_p(p(\theta), q(\theta)) \tag{6.56}$$

where $\lambda > 0$ is a regularization hyperparameter, and $\text{KL}(q(\theta) \parallel p(\theta))$ is the Kullback-Leibler divergence. The Wasserstein term $W_p(p(\theta), q(\theta))$ enforces geometric consistency between the prior and the posterior, ensuring that the posterior does not deviate excessively from the prior in a manner that respects the underlying metric structure of Θ. The smoothness of the posterior is guaranteed by the properties of the Wasserstein distance. Specifically, the Wasserstein distance is Lipschitz-continuous with respect to perturbations in the distributions. For any $P, Q, R \in P_p(X)$, the triangle inequality holds:

$$W_p(P,Q) \leq W_p(P,R) + W_p(R,Q) \tag{6.57}$$

This inequality ensures that small changes in the data distribution D or the prior $p(\theta)$ lead to proportionally small changes in the posterior $q(\theta)$, thereby promoting stability. Furthermore, the Wasserstein distance is convex in its arguments, which facilitates efficient optimization and guarantees the existence of a unique minimizer under appropriate conditions. To compute the Wasserstein distance in practice, we often employ entropic regularization, which transforms the OT problem into a strictly convex optimization problem. The entropic regularized Wasserstein distance $W_{p,\epsilon}(P, Q)$ is defined as:

$$W_{p,\epsilon}(P,Q) = \inf_{\pi \in \Pi(P,Q)} \int_{X \times X} d(x,y)^p \, d\pi(x,y) + \epsilon H(\pi) \tag{6.58}$$

where $H(\pi) = \int_{X\times X} \pi(x,y)\log\pi(x,y)\,dx\,dy$ is the entropy of the coupling π, and $\epsilon > 0$ is the regularization parameter. The entropic regularization ensures that the optimal coupling π^* is unique and has the form:

$$\pi^*(x,y) = \exp\left(\frac{f(x)+g(y)-d(x,y)^p}{\epsilon}\right) \tag{6.59}$$

where f and g are dual potentials that satisfy the Schrödinger system of equations. This formulation allows for efficient computation using the Sinkhorn-Knopp algorithm, which iteratively updates the dual potentials f and g. The stability and smoothness of the posterior are further reinforced by the dual formulation of the Wasserstein distance. By the Kantorovich-Rubinstein duality, the 1-Wasserstein distance can be expressed as:

$$W_1(P,Q) = \sup_{f\in \mathrm{Lip}_1(X)} \left| \int_X f(x)\,dP(x) - \int_X f(x)\,dQ(x) \right| \tag{6.60}$$

where $\mathrm{Lip}_1(X)$ is the set of 1-Lipschitz functions on X. This dual formulation highlights the role of Lipschitz continuity in controlling the smoothness of the posterior, as the Wasserstein distance penalizes functions that vary too rapidly.

In summary, OT-based priors ensure that posterior distributions maintain smooth and stable representations by leveraging the geometric and analytical properties of the Wasserstein distance. The Wasserstein distance provides a rigorous framework for regularizing Bayesian inference, ensuring that the posterior remains close to the prior in a geometrically meaningful way. This approach is computationally tractable due to entropic regularization and the Sinkhorn-Knopp algorithm, and it is theoretically grounded in measure theory, convex analysis, and functional analysis. The resulting Bayesian framework is robust to noise and distributional shifts, leading to improved generalization in uncertainty estimation tasks.

6.4 Application of Optimal Transport in learning energy-based models

The foundational principle of Optimal Transport lies in the minimization of the cost required to transport mass from one probability distribution to another. Let (X,d) be a metric space, where X is the sample space and $d : X\times X \to \mathbb{R}^+$ is a metric. Given two probability measures P_{data} and P_θ defined on X, the OT problem seeks to find the optimal coupling γ that minimizes the total transportation cost:

$$W_c(P_{\text{data}}, P_\theta) = \inf_{\gamma\in\Gamma(P_{\text{data}},P_\theta)} \int_{X\times X} c(x,y)\,d\gamma(x,y), \tag{6.61}$$

where $\Gamma(P_{\text{data}}, P_\theta)$ denotes the set of all joint probability measures γ on $X\times X$ with marginals P_{data} and P_θ, and $c(x,y)$ is a cost function, typically chosen as $c(x,y) = d(x,y)^p$ for $p \geq 1$. For $p = 2$, this yields the squared Euclidean cost, and the corresponding OT distance is the 2-Wasserstein distance:

$$W_2(P_{\text{data}}, P_\theta) = \left(\inf_{\gamma\in\Gamma(P_{\text{data}},P_\theta)} \int_{X\times X} \|x-y\|_2^2\,d\gamma(x,y) \right)^{1/2}. \tag{6.62}$$

In the context of energy-based models, the model distribution $P_\theta(x)$ is defined via an energy function $E_\theta(x)$ and the Boltzmann-Gibbs distribution:

$$P_\theta(x) = \frac{1}{Z(\theta)}\exp(-E_\theta(x)), \tag{6.63}$$

$$Z(\theta) = \int_X \exp(-E_\theta(x))\,dx, \tag{6.64}$$

where $Z(\theta)$ is the partition function. The goal is to minimize the discrepancy between P_{data} and P_θ, which can be achieved by minimizing the Wasserstein distance $W_2(P_{\text{data}}, P_\theta)$. The Kantorovich duality theorem provides a dual formulation of the OT problem:

$$W_2^2(P_{\text{data}}, P_\theta) = \sup_{\varphi,\psi} \left\{ \int_X \varphi(x)\, dP_{\text{data}}(x) + \int_X \psi(y)\, dP_\theta(y) \right\}, \tag{6.65}$$

subject to the constraint $\varphi(x)+\psi(y) \leq \|x-y\|_2$ for all $x, y \in X$. Here, φ and ψ are the Kantorovich potentials, which are related via the c-transform:

$$\varphi_c(y) = \inf_{x\in X} \left\{ \|x-y\|_2 - \varphi(x) \right\}, \tag{6.66}$$

$$\psi_c(x) = \inf_{y\in X} \left\{ \|x-y\|_2 - \psi(y) \right\}. \tag{6.67}$$

The Kantorovich potentials φ and ψ can be parameterized using neural networks, leading to the Wasserstein Generative Adversarial Network (WGAN) framework. In this setting, the discriminator learns φ, while the generator minimizes the Wasserstein distance. The gradient of the Wasserstein distance with respect to the model parameters θ is given by:

$$\nabla_\theta W_2^2(P_{\text{data}}, P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}} \left[\nabla_\theta \varphi_\theta(x) \right] - \mathbb{E}_{y\sim P_\theta} \left[\nabla_\theta \varphi_\theta(y) \right], \tag{6.68}$$

where φ_θ is the Kantorovich potential parameterized by θ. This gradient can be estimated using Monte Carlo sampling from P_{data} and P_θ, enabling stochastic optimization. To enhance computational efficiency and stability, entropy regularization is often introduced, leading to the Sinkhorn divergence:

$$W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) = \inf_{\gamma\in\Gamma(P_{\text{data}}, P_\theta)} \left\{ \int_{X\times X} \|x-y\|_2^2\, d\gamma(x,y) + \epsilon H(\gamma) \right\}, \tag{6.69}$$

where

$$H(\gamma) = \int_{X\times X} \gamma(x,y) \log \gamma(x,y)\, dx\, dy \tag{6.70}$$

is the entropy of the coupling γ, and $\epsilon > 0$ is the regularization parameter. The Sinkhorn divergence provides a smoothed approximation of the Wasserstein distance, which is particularly advantageous in high-dimensional settings. The optimization problem for learning EBMs using OT can be formulated as:

$$\min_\theta W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) + \lambda R(\theta), \tag{6.71}$$

where $R(\theta)$ is a regularization term, such as $\|\theta\|_2^2$, and λ is a hyperparameter. The gradient of the Sinkhorn divergence with respect to θ is:

$$\nabla_\theta W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}} \left[\nabla_\theta \varphi_\theta(x) \right] - \mathbb{E}_{y\sim P_\theta} \left[\nabla_\theta \varphi_\theta(y) \right] + \epsilon \nabla_\theta H(\gamma_\theta), \tag{6.72}$$

where γ_θ is the optimal coupling under entropy regularization. This gradient can be computed efficiently using the Sinkhorn-Knopp algorithm, which iteratively updates the coupling γ and the potentials φ.

In summary, the application of Optimal Transport to learning energy-based models provides a mathematically rigorous framework for minimizing the discrepancy between data and model distributions. The interplay between Kantorovich duality, entropy regularization, and stochastic optimization enables efficient and scalable learning of complex, high-dimensional distributions. The resulting OT-based EBMs are both theoretically sound and practically effective, with connections to generative models such as WGANs and variational inference.

6.5 Sinkhorn-Knopp Algorithm

The Sinkhorn-Knopp algorithm is a numerical method for solving entropy-regularized optimal transport problems, which are central to many applications in deep neural networks, including generative modeling, domain adaptation, and representation learning. The optimal transport problem seeks to find a coupling matrix $P \in \mathbb{R}_+^{n\times m}$ that minimizes the transportation cost between two probability distributions $\mathbf{a} \in \mathbb{R}_+^n$ and $\mathbf{b} \in \mathbb{R}_+^m$, subject to marginal constraints. The cost is quantified by a cost matrix $C \in \mathbb{R}_+^{n\times m}$, where C_{ij} represents the cost of transporting mass from point i in the source distribution to point j in the target distribution. The entropy-regularized optimal transport problem introduces a regularization term to the objective function, resulting in the following convex optimization problem:

$$\min_{P\in\mathbb{R}_+^{n\times m}} \langle P, C\rangle - \epsilon H(P), \tag{6.73}$$

subject to the constraints:

$$P\mathbf{1}_m = \mathbf{a}, \quad P^\top \mathbf{1}_n = \mathbf{b}, \tag{6.74}$$

where $\langle P, C\rangle = \sum_{i=1}^n \sum_{j=1}^m P_{ij}C_{ij}$ is the Frobenius inner product, $\epsilon > 0$ is the regularization parameter, and $H(P) = -\sum_{i=1}^n \sum_{j=1}^m P_{ij}(\log P_{ij} - 1)$ is the entropy of the coupling matrix P. The regularization term $\epsilon H(P)$ ensures that the solution P is smooth and computationally tractable, as it penalizes deviations from uniformity. The Sinkhorn-Knopp algorithm solves this problem by leveraging the dual formulation of the entropy-regularized optimal transport problem. The dual problem can be derived using the method of Lagrange multipliers, introducing dual variables $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^m$ corresponding to the marginal constraints. The Lagrangian of the problem is given by:

$$\mathcal{L}(P, \mathbf{f}, \mathbf{g}) = \langle P, C\rangle - \epsilon H(P) - \mathbf{f}^\top(P\mathbf{1}_m - \mathbf{a}) - \mathbf{g}^\top(P^\top\mathbf{1}_n - \mathbf{b}). \tag{6.75}$$

Taking the gradient of the Lagrangian with respect to P and setting it to zero yields the optimality condition:

$$P_{ij} = \exp\left(\frac{f_i + g_j - C_{ij}}{\epsilon}\right). \tag{6.76}$$

This expression can be rewritten in terms of the Gibbs kernel $K \in \mathbb{R}_+^{n\times m}$, defined as $K_{ij} = \exp(-C_{ij}/\epsilon)$, and the scaling vectors $\mathbf{u} = \exp(\mathbf{f}/\epsilon)$ and $\mathbf{v} = \exp(\mathbf{g}/\epsilon)$. The coupling matrix P is then given by:

$$P_{ij} = u_i K_{ij} v_j. \tag{6.77}$$

The Sinkhorn-Knopp algorithm iteratively updates the scaling vectors $\mathbf{u}$ and $\mathbf{v}$ to satisfy the marginal constraints $P\mathbf{1}_m = \mathbf{a}$ and $P^\top\mathbf{1}_n = \mathbf{b}$. The update rules are derived from the optimality conditions and are given by:

$$u_i^{(k+1)} = \frac{a_i}{\sum_{j=1}^m K_{ij} v_j^{(k)}}, \quad v_j^{(k+1)} = \frac{b_j}{\sum_{i=1}^n K_{ij} u_i^{(k+1)}} \tag{6.78}$$

These updates ensure that the coupling matrix P converges to the unique solution of the entropy-regularized optimal transport problem. The convergence of the algorithm is guaranteed by the contractive nature of the updates, and the rate of convergence is linear in the number of iterations. In the context of deep neural networks, the Sinkhorn-Knopp algorithm is often used to compute approximate Wasserstein distances, which are essential for training generative models such as Wasserstein GANs. The Wasserstein distance between two probability distributions $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$W(\mathbf{a}, \mathbf{b}) = \min_{P\in\Pi(\mathbf{a},\mathbf{b})} \langle P, C\rangle, \tag{6.79}$$

where $\Pi(\mathbf{a}, \mathbf{b})$ is the set of all coupling matrices satisfying the marginal constraints. The entropy-regularized version of the Wasserstein distance, known as the Sinkhorn distance, is given by:

$$W_\epsilon(\mathbf{a}, \mathbf{b}) = \langle P^*, C\rangle - \epsilon H(P^*), \tag{6.80}$$

where P^* is the solution to the entropy-regularized optimal transport problem. The Sinkhorn-Knopp algorithm provides an efficient way to compute P^* and, consequently, the Sinkhorn distance. The algorithm can also be extended to handle unbalanced optimal transport problems, where the source and target distributions do not necessarily have the same total mass. In this case, the optimization problem is modified to include additional penalty terms for deviations from the marginal constraints:

$$\min_{P \in \mathbb{R}_+^{n\times m}} \langle P, C\rangle - \epsilon H(P) + \lambda_1 \mathrm{KL}(P\mathbf{1}_m \| \mathbf{a}) + \lambda_2 \mathrm{KL}(P^\top \mathbf{1}_n \| \mathbf{b}), \tag{6.81}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence, and $\lambda_1, \lambda_2 > 0$ are regularization parameters. The Sinkhorn-Knopp algorithm can be adapted to solve this problem by modifying the update rules for $\mathbf{u}$ and $\mathbf{v}$:

$$u_i^{(k+1)} = \frac{a_i}{\sum_{j=1}^m K_{ij} v_j^{(k)}} + \lambda_1, \quad v_j^{(k+1)} = \frac{b_j}{\sum_{i=1}^n K_{ij} u_i^{(k+1)}} + \lambda_2 \tag{6.82}$$

This extension allows the algorithm to handle a broader range of applications in deep learning, such as semi-supervised learning and data augmentation, where the source and target distributions may not be perfectly aligned.

In summary, the Sinkhorn-Knopp algorithm is a highly efficient and mathematically rigorous method for solving entropy-regularized optimal transport problems. Its iterative nature, linear convergence rate, and ability to handle both balanced and unbalanced transport problems make it a versatile tool in deep learning. The algorithm's reliance on matrix scaling operations ensures that it can be seamlessly integrated into existing deep learning frameworks, making it a cornerstone of modern optimal transport theory and its applications.

6.5.1 Sinkhorn Distance

The Sinkhorn distance is a mathematically rigorous and computationally efficient approximation of the optimal transport problem, which seeks to minimize the cost of transporting mass from one probability distribution to another. Let $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ be two discrete probability distributions defined over finite spaces, and let $\mathbf{C} \in \mathbb{R}^{n\times m}$ be a cost matrix where C_{ij} represents the cost of transporting a unit of mass from point i in the support of $\mathbf{a}$ to point j in the support of $\mathbf{b}$. The optimal transport problem is formulated as:

$$\mathrm{OT}(\mathbf{a}, \mathbf{b}) = \min_{\mathbf{P} \in \mathbf{U}(\mathbf{a},\mathbf{b})} \langle \mathbf{P}, \mathbf{C}\rangle, \tag{6.83}$$

where $\mathbf{U}(\mathbf{a}, \mathbf{b}) = \{\mathbf{P} \in \mathbb{R}_+^{n\times m} \mid \mathbf{P}\mathbf{1}_m = \mathbf{a}, \mathbf{P}^\top \mathbf{1}_n = \mathbf{b}\}$ is the set of all valid transport plans, $\mathbf{1}_n$ and $\mathbf{1}_m$ are vectors of ones, and $\langle \mathbf{P}, \mathbf{C}\rangle = \sum_{i=1}^n \sum_{j=1}^m P_{ij} C_{ij}$ is the Frobenius inner product. The Sinkhorn distance introduces an entropic regularization term to this problem, resulting in the following optimization problem:

$$\mathrm{OT}_\lambda(\mathbf{a}, \mathbf{b}) = \min_{\mathbf{P} \in \mathbf{U}(\mathbf{a},\mathbf{b})} \langle \mathbf{P}, \mathbf{C}\rangle - \lambda H(\mathbf{P}), \tag{6.84}$$

where $\lambda > 0$ is the regularization parameter, and $H(\mathbf{P}) = -\sum_{i=1}^n \sum_{j=1}^m P_{ij}(\log P_{ij} - 1)$ is the entropy of the transport plan $\mathbf{P}$. The entropic regularization ensures that the optimal transport plan $\mathbf{P}^*$ is unique and has the form:

$$\mathbf{P}^* = \mathrm{diag}(\mathbf{u})\mathbf{K}\mathrm{diag}(\mathbf{v}), \tag{6.85}$$

where $\mathbf{K} = \exp(-\mathbf{C}/\lambda)$ is the Gibbs kernel, and $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^m$ are positive scaling vectors that satisfy the marginal constraints:

$$\mathbf{u} \odot (\mathbf{K}\mathbf{v}) = \mathbf{a} \quad \text{and} \quad \mathbf{v} \odot (\mathbf{K}^\top \mathbf{u}) = \mathbf{b}. \tag{6.86}$$

Here, $\odot$ denotes element-wise multiplication. The Sinkhorn-Knopp algorithm is used to iteratively compute these scaling vectors:

$$\mathbf{u}^{(k+1)} = \frac{\mathbf{a}}{\mathbf{K}\mathbf{v}^{(k)}} \quad \text{and} \quad \mathbf{v}^{(k+1)} = \frac{\mathbf{b}}{\mathbf{K}^\top \mathbf{u}^{(k+1)}}, \tag{6.87}$$

where $\mathbf{u}^{(0)} = \mathbf{1}_n$ and $\mathbf{v}^{(0)} = \mathbf{1}_m$ are initialized as vectors of ones. The algorithm converges to the optimal scaling vectors $\mathbf{u}^*$ and $\mathbf{v}^*$, which define the optimal transport plan $\mathbf{P}^*$. The Sinkhorn distance is then given by:

$$\text{Sinkhorn}_\lambda(\mathbf{a}, \mathbf{b}) = \langle \mathbf{P}^*, \mathbf{C} \rangle - \lambda H(\mathbf{P}^*). \tag{6.88}$$

The regularization parameter λ controls the trade-off between the transportation cost and the entropy of the transport plan. As $\lambda \to \infty$, the Sinkhorn distance approaches the original optimal transport distance, while as $\lambda \to 0$, the transport plan becomes more diffuse and the distance approaches the entropy-regularized Wasserstein distance. The Sinkhorn distance is computationally efficient due to the iterative nature of the Sinkhorn-Knopp algorithm, making it suitable for large-scale applications. The dual formulation of the regularized optimal transport problem provides additional insights into the structure of the Sinkhorn distance. The dual problem is given by:

$$\text{OT}_\lambda(\mathbf{a}, \mathbf{b}) = \max_{\mathbf{f}, \mathbf{g}} \mathbf{f}^\top \mathbf{a} + \mathbf{g}^\top \mathbf{b} - \lambda \sum_{i=1}^n \sum_{j=1}^m \exp\left(\frac{f_i + g_j - C_{ij}}{\lambda}\right), \tag{6.89}$$

where $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^m$ are dual variables. The optimal dual variables $\mathbf{f}^*$ and $\mathbf{g}^*$ are related to the scaling vectors $\mathbf{u}^*$ and $\mathbf{v}^*$ by:

$$\mathbf{u}^* = \exp\left(\frac{\mathbf{f}^*}{\lambda}\right) \quad \text{and} \quad \mathbf{v}^* = \exp\left(\frac{\mathbf{g}^*}{\lambda}\right). \tag{6.90}$$

The Sinkhorn distance can thus be computed using either the primal or dual formulation, depending on the specific application. The dual formulation is particularly useful for deriving theoretical properties of the Sinkhorn distance, such as its convexity and smoothness with respect to the input distributions $\mathbf{a}$ and $\mathbf{b}$. The Sinkhorn distance satisfies several key mathematical properties. First, it is symmetric, meaning that $\text{Sinkhorn}_\lambda(\mathbf{a}, \mathbf{b}) = \text{Sinkhorn}_\lambda(\mathbf{b}, \mathbf{a})$. Second, it satisfies the triangle inequality, making it a valid metric on the space of probability distributions. Third, it is differentiable with respect to the input distributions $\mathbf{a}$ and $\mathbf{b}$, which is useful for gradient-based optimization in machine learning applications. Finally, the Sinkhorn distance is computationally efficient, with a time complexity of $O(nm)$ per iteration of the Sinkhorn-Knopp algorithm, making it scalable to high-dimensional problems. The Sinkhorn distance can also be generalized to continuous probability measures using the Kantorovich formulation of optimal transport. Let μ and ν be probability measures defined on metric spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, and let $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ be a continuous cost function. The entropically regularized optimal transport problem in the continuous setting is given by:

$$\text{OT}_\lambda(\mu, \nu) = \min_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) \, d\pi(x, y) - \lambda H(\pi), \tag{6.91}$$

where $\Pi(\mu, \nu)$ is the set of all joint probability measures π with marginals μ and ν, and $H(\pi) = -\int_{\mathcal{X} \times \mathcal{Y}} \log\left(\frac{d\pi}{d\mu \otimes d\nu}\right) d\pi$ is the entropy of the coupling π. The optimal coupling π^* has the form:

$$d\pi^*(x, y) = \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda}\right) d\mu(x) d\nu(y), \tag{6.92}$$

where f and g are continuous functions satisfying the Schrödinger system:

$$\int_{\mathcal{Y}} \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda}\right) d\nu(y) = 1 \quad \text{and} \quad \int_{\mathcal{X}} \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda}\right) d\mu(x) = 1. \tag{6.93}$$

The Sinkhorn distance in the continuous setting is then given by:

$$\text{Sinkhorn}_\lambda(\mu,\nu) = \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\, d\pi^*(x,y) - \lambda H(\pi^*). \tag{6.94}$$

This formulation extends the Sinkhorn distance to continuous probability measures, enabling its application to a broader class of problems in probability theory, statistics, and machine learning. The Sinkhorn distance thus provides a rigorous and computationally efficient framework for measuring the dissimilarity between probability distributions, combining the theoretical foundations of optimal transport with the practical advantages of entropic regularization.

6.5.2 Wasserstein GANs

Wasserstein Generative Adversarial Networks (WGANs) are a class of generative models that utilize the Wasserstein-1 distance, also known as the Earth Mover's Distance (EMD), to measure the discrepancy between the real data distribution $\mathbb{P}_r$ and the generated data distribution $\mathbb{P}_g$. The Wasserstein-1 distance is defined as:

$$W_1(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{6.95}$$

Here, $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ represents the set of all joint distributions $\gamma(x,y)$ with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$, and the infimum is taken over all such joint distributions. The Wasserstein-1 distance quantifies the minimal cost required to transport mass from $\mathbb{P}_r$ to $\mathbb{P}_g$, where the cost is proportional to the Euclidean distance $\|x-y\|$. In WGANs, the generator G and the critic D are optimized to minimize and maximize the Wasserstein-1 distance, respectively. The critic D is trained to approximate the Wasserstein-1 distance between $\mathbb{P}_r$ and $\mathbb{P}_g$. The objective function for the critic is given by:

$$L_D = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))] \tag{6.96}$$

Here, z is a random noise vector sampled from a prior distribution $p_z(z)$, and $G(z)$ is the generated sample. To ensure that the critic D is a 1-Lipschitz function, a constraint is imposed on its gradients. This constraint is typically enforced using a gradient penalty term, leading to the modified critic loss function:

$$L_D^{GP} = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2] \tag{6.97}$$

In this equation, λ is a hyperparameter controlling the strength of the gradient penalty, and $\hat{x}$ is sampled uniformly along straight lines connecting pairs of points sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. The term $(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2$ penalizes deviations from the 1-Lipschitz constraint, ensuring that the critic's gradients have a norm of at most 1. The generator G is trained to minimize the Wasserstein-1 distance by maximizing the critic's output on generated samples. The objective function for the generator is given by:

$$L_G = -\mathbb{E}_{z\sim p_z(z)}[D(G(z))] \tag{6.98}$$

The optimization process alternates between updating the critic and the generator. The critic is updated multiple times for each update of the generator to ensure that it remains close to the optimal 1-Lipschitz function. The parameter updates for the critic and generator are given by:

$$\theta_D \leftarrow \theta_D - \eta_D\nabla_{\theta_D}L_D^{GP} \tag{6.99}$$

$$\theta_G \leftarrow \theta_G - \eta_G\nabla_{\theta_G}L_G \tag{6.100}$$

Here, θ_D and θ_G are the parameters of the critic and generator, respectively, and η_D and η_G are the learning rates for the critic and generator. The training continues until the generator produces samples that are indistinguishable from real data according to the critic. The Wasserstein-1 distance provides several theoretical advantages over traditional GAN objectives, such as the

Jensen-Shannon divergence used in the original GAN formulation. The Wasserstein-1 distance is continuous and differentiable almost everywhere, which leads to more stable training dynamics. Additionally, the Wasserstein-1 distance correlates better with sample quality, providing a meaningful metric for evaluating the performance of the generator. The mathematical properties of the Wasserstein-1 distance, including its sensitivity to the geometry of the underlying space and its ability to provide meaningful gradients even when the supports of $\mathbb{P}_r$ and $\mathbb{P}_g$ are disjoint, make WGANs a powerful tool for generative modeling.

In summary, WGANs optimize the following minimax objective:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \tag{6.101}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. The Wasserstein-1 distance $W_1(\mathbb{P}_r, \mathbb{P}_g)$ is approximated by the critic D, and the generator G is trained to minimize this distance. The use of the Wasserstein-1 distance in WGANs provides a theoretically grounded framework for generative modeling, leading to improved stability and performance compared to traditional GANs. The mathematical rigor of the Wasserstein-1 distance and its properties ensure that WGANs are well-suited for a wide range of generative tasks.

6.6 Kantorovich Duality

The Kantorovich duality theorem is a fundamental result in optimal transport theory that establishes the equivalence between the primal formulation of the optimal transport problem and a dual formulation based on potential functions. Given two probability spaces $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \nu)$, with a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, the primal Kantorovich formulation of optimal transport seeks to minimize the total transport cost over all possible couplings γ of μ and ν, where a coupling is a joint probability measure on $\mathcal{X} \times \mathcal{Y}$ whose marginals are μ and ν, respectively. Mathematically, this is expressed as

$$\inf_{\gamma \in \Pi(\mu,\nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y)\, d\gamma(x, y), \tag{6.102}$$

where $\Pi(\mu, \nu)$ is the set of all couplings satisfying

$$\int_{\mathcal{X}} d\gamma(x, y) = d\nu(y), \quad \int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x). \tag{6.103}$$

The Kantorovich duality states that the optimal transport cost can equivalently be expressed in terms of potential functions $\varphi : \mathcal{X} \to \mathbb{R}$ and $\psi : \mathcal{Y} \to \mathbb{R}$ satisfying the inequality

$$\varphi(x) + \psi(y) \leq c(x, y), \quad \forall (x, y) \in \mathcal{X} \times \mathcal{Y}. \tag{6.104}$$

Under appropriate conditions, the optimal transport cost is given by the dual formulation

$$\sup_{\varphi,\psi} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) \right\} \tag{6.105}$$

subject to the constraint

$$\varphi(x) + \psi(y) \leq c(x, y), \quad \forall (x, y). \tag{6.106}$$

A key result in this formulation is that when the cost function $c(x, y)$ is lower semi-continuous, the supremum is attained by functions φ and ψ that satisfy

$$\psi(y) = \inf_{x \in \mathcal{X}} \left[c(x, y) - \varphi(x) \right]. \tag{6.107}$$

For the specific case of the Wasserstein-1 distance, where $c(x, y) = \|x - y\|$, the dual problem simplifies to

$$W_1(\mu, \nu) = \sup_{\|\varphi\|_L \leq 1} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) - \int_{\mathcal{Y}} \varphi(y) d\nu(y) \right\}. \tag{6.108}$$

Here, the supremum is taken over all 1-Lipschitz functions $\varphi : \mathcal{X} \to \mathbb{R}$, satisfying

$$|\varphi(x) - \varphi(y)| \leq \|x - y\|, \quad \forall x, y \in \mathcal{X}. \tag{6.109}$$

This dual formulation is of fundamental importance in deep learning applications, particularly in the training of Wasserstein Generative Adversarial Networks (WGANs). In this setting, the generator $G : \mathcal{Z} \to \mathcal{X}$ transforms samples $z \sim p_z$ from a latent space $\mathcal{Z}$ into synthetic data points in $\mathcal{X}$, aiming to approximate a target data distribution μ. The discriminator network approximates the Kantorovich potential φ, leading to the objective

$$\min_G \max_{\|\varphi\|_L \leq 1} \left\{ \mathbb{E}_{x \sim \mu}[\varphi(x)] - \mathbb{E}_{y \sim \nu}[\varphi(G(z))] \right\}. \tag{6.110}$$

This adversarial framework provides a more stable training procedure than standard GANs, which rely on the Jensen-Shannon divergence. The Kantorovich duality ensures that the discriminator function learns a transport potential that characterizes the optimal transport map, guiding the generator toward minimizing the Wasserstein distance between the real and generated distributions. Beyond generative modeling, optimal transport theory is also applied in supervised learning, where the Wasserstein distance serves as a loss function for comparing distributions in high-dimensional spaces. When computational efficiency is required, entropic regularization techniques introduce an additional entropy penalty term $H(\gamma)$, yielding the regularized transport problem

$$\inf_{\gamma \in \Pi(\mu, \nu)} \left\{ \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) d\gamma(x, y) + \epsilon H(\gamma) \right\}, \tag{6.111}$$

where

$$H(\gamma) = \int_{\mathcal{X} \times \mathcal{Y}} \gamma(x, y) \log \gamma(x, y) dx dy. \tag{6.112}$$

This modification allows for efficient computation via Sinkhorn iterations, leveraging the dual formulation

$$\sup_{\varphi, \psi} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) - \epsilon \int_{\mathcal{X} \times \mathcal{Y}} e^{(\varphi(x) + \psi(y) - c(x, y))/\epsilon} dx dy \right\}. \tag{6.113}$$

Such approaches are particularly useful in deep learning, where high-dimensional transport problems arise in applications such as domain adaptation, clustering, and metric learning. The theoretical underpinnings of Kantorovich duality thus play a critical role in designing robust algorithms for optimizing probability distributions in high-dimensional spaces, ensuring efficient convergence and improved generalization performance in deep neural networks.

6.7 Entropy Regularization

The entropy regularization of optimal transport arises from the classical Monge-Kantorovich formulation, where the goal is to find a transport plan $\gamma \in \Pi(\mu, \nu)$ that minimizes the expected transportation cost between two probability measures μ and ν. Mathematically, this can be expressed as the minimization problem

$$W_c(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) \, d\gamma(x, y), \tag{6.114}$$

where $\Pi(\mu, \nu)$ is the space of joint probability distributions on $\mathcal{X} \times \mathcal{Y}$ with marginals μ and ν, respectively, satisfying the marginalization conditions

$$\int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x), \quad \int_{\mathcal{X}} d\gamma(x, y) = d\nu(y). \tag{6.115}$$

While this optimization problem is well-posed, it is computationally intractable in high-dimensional settings due to the combinatorial explosion of feasible transport plans. To mitigate this, an entropy regularization term is introduced, leading to the entropy-regularized optimal transport problem

$$W_c^\lambda(\mu, \nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \left(\int_{\mathcal{X}\times\mathcal{Y}} c(x, y) d\gamma(x, y) + \lambda H(\gamma) \right), \tag{6.116}$$

where $H(\gamma)$ is the Shannon entropy of the transport plan γ, given by

$$H(\gamma) = -\int_{\mathcal{X}\times\mathcal{Y}} \gamma(x, y) \log \gamma(x, y)\, dx\, dy. \tag{6.117}$$

The introduction of entropy regularization has fundamental mathematical consequences. First, it ensures the strict convexity of the optimization problem, which allows for efficient computation using iterative methods. Second, it guarantees the uniqueness of the optimal transport plan, preventing the degeneracy issues that arise in the classical unregularized setting. The optimal transport plan in the entropy-regularized setting takes the form

$$\gamma^\lambda(x, y) = \exp\left(-\frac{c(x, y)}{\lambda}\right) \mu(x)\nu(y), \tag{6.118}$$

which is known as the Gibbs kernel representation. This form follows from the variational principle associated with entropy maximization under marginal constraints, leading to the closed-form solution

$$\gamma^\lambda(x, y) = \mu(x)\nu(y)e^{\frac{\phi(x)+\psi(y)-c(x,y)}{\lambda}}, \tag{6.119}$$

where $\phi(x)$ and $\psi(y)$ are dual potentials arising from the Lagrange multipliers enforcing the marginal constraints. The dual formulation of entropy-regularized optimal transport is derived from the Fenchel-Rockafellar duality theorem and takes the form

$$\sup_{\phi,\psi} \left(\int_{\mathcal{X}} \phi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) - \lambda \int_{\mathcal{X}\times\mathcal{Y}} e^{\frac{\phi(x)+\psi(y)-c(x,y)}{\lambda}} dxdy \right). \tag{6.120}$$

The dual potentials $\phi(x)$ and $\psi(y)$ satisfy the iterative update equations

$$\phi^{(k+1)}(x) = -\lambda \log \int_{\mathcal{Y}} e^{\frac{\psi^{(k)}(y)-c(x,y)}{\lambda}} d\nu(y), \tag{6.121}$$

$$\psi^{(k+1)}(y) = -\lambda \log \int_{\mathcal{X}} e^{\frac{\phi^{(k)}(x)-c(x,y)}{\lambda}} d\mu(x). \tag{6.122}$$

These updates converge exponentially to the optimal dual potentials, enabling the efficient computation of entropy-regularized transport distances. In the discrete setting, where the probability measures μ and ν are represented as vectors $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^m$, the entropy-regularized optimal transport problem reduces to a matrix scaling problem. Given a cost matrix $C \in \mathbb{R}^{n\times m}$, the optimal transport matrix T satisfies

$$T_{ij} = \exp\left(-\frac{C_{ij}}{\lambda}\right) u_i v_j. \tag{6.123}$$

This leads to the Sinkhorn-Knopp iteration

$$u^{(k+1)} = \frac{\mu}{Kv^{(k)}}, \quad v^{(k+1)} = \frac{\nu}{K^T u^{(k)}}, \tag{6.124}$$

where $K_{ij} = e^{-C_{ij}/\lambda}$ and division is element-wise. These updates ensure fast convergence and numerical stability, making entropy-regularized optimal transport suitable for large-scale applications in deep learning. In the context of deep neural networks, entropy-regularized optimal transport plays a crucial role in generative modeling, domain adaptation, and adversarial training. Given a parameterized generator $G_\theta(z)$ with latent variable $z \sim P_Z$, the entropy-regularized Wasserstein loss for training a generative model is given by

$$\mathcal{L}(\theta) = W_c^\lambda(P_{G_\theta}, P_X) = \inf_{\gamma \in \Pi(P_{G_\theta}, P_X)} \left(\int c(G_\theta(z), x) d\gamma(z, x) + \lambda H(\gamma) \right). \tag{6.125}$$

This formulation enables stable gradient-based training by ensuring smoothness in the transport plan. Furthermore, in domain adaptation, the entropy-regularized Wasserstein distance is minimized between the source and target feature distributions P_S and P_T, leading to the optimization problem

$$\mathcal{L}_{\text{OT}} = W_c^\lambda(P_S, P_T) = \inf_{\gamma \in \Pi(P_S, P_T)} \left(\int c(x, y) d\gamma(x, y) + \lambda H(\gamma) \right). \tag{6.126}$$

This objective ensures that the learned representation preserves geometric consistency between domains while mitigating domain shift. Thus, entropy regularization in optimal transport provides a mathematically rigorous, computationally efficient, and theoretically well-grounded framework for deep learning applications, ensuring convexity, uniqueness, and numerical stability while enabling large-scale optimization in high-dimensional probability spaces.

6.7.1 Classical Monge-Kantorovich formulation

The **Monge-Kantorovich optimal transport problem** seeks to determine the most efficient way to transport mass from one distribution to another while minimizing a given cost function. Let X and Y be two measurable spaces, each endowed with probability measures μ and ν, respectively, which are absolutely continuous with respect to the Lebesgue measure, having corresponding density functions $f(x)$ and $g(y)$. The fundamental problem first considered by Monge is to find a measurable transport map $T : X \to Y$ satisfying the **pushforward condition**,

$$T_\# \mu = \nu, \quad \text{or equivalently,} \quad \int_A g(y) dy = \int_{T^{-1}(A)} f(x) dx, \quad \forall A \subset Y, \tag{6.127}$$

which ensures that mass is conserved. Given a cost function $c : X \times Y \to \mathbb{R}$, the total transport cost associated with the map T is given by the functional

$$\mathcal{C}[T] = \int_X c(x, T(x)) f(x) dx. \tag{6.128}$$

The **Monge problem** is then formulated as the minimization problem

$$\inf_T \mathcal{C}[T] = \inf_T \int_X c(x, T(x)) f(x) dx, \tag{6.129}$$

where the infimum is taken over all measurable transport maps satisfying the pushforward condition. The primary difficulty in solving Monge's problem lies in the fact that a transport map may not exist in general, particularly when the measure μ is more **spread out** than ν, or when mass-splitting is required, which motivates the **relaxation** of Monge's problem to the Kantorovich formulation. Instead of restricting attention to deterministic transport maps, Kantorovich introduced the notion of **transport plans**, which are probability measures γ on the product space $X \times Y$ satisfying the **marginal constraints**

$$\int_Y d\gamma(x, y) = d\mu(x), \quad \int_X d\gamma(x, y) = d\nu(y). \tag{6.130}$$

The set of all such **couplings** is denoted as $\Pi(\mu, \nu)$, which consists of all probability measures $\gamma(x, y)$ whose projections onto X and Y coincide with μ and ν, respectively. The **Kantorovich optimal transport problem** is then formulated as the minimization problem

$$\inf_{\gamma \in \Pi(\mu,\nu)} \int_{X \times Y} c(x, y) d\gamma(x, y). \tag{6.131}$$

Under mild conditions, such as compactness of X and Y and continuity of $c(x, y)$, existence of an **optimal transport plan** γ^* is guaranteed. If γ^* is supported on the graph of a function $y = T(x)$, then the optimal plan corresponds to a Monge-type transport. The problem of characterizing such solutions leads naturally to **Kantorovich duality**, which provides a variational formulation of optimal transport in terms of potentials $\varphi : X \to \mathbb{R}$ and $\psi : Y \to \mathbb{R}$. The dual problem is expressed as

$$\sup_{\varphi,\psi} \left\{ \int_X \varphi(x) d\mu(x) + \int_Y \psi(y) d\nu(y) \right\} \tag{6.132}$$

subject to the inequality constraint

$$\varphi(x) + \psi(y) \leq c(x, y), \quad \forall (x, y) \in X \times Y. \tag{6.133}$$

The optimality conditions for the Kantorovich dual problem imply the **complementary slackness condition**

$$\varphi(x) + \psi(y) = c(x, y) \quad \text{for} \quad (x, y) \in \operatorname{supp}(\gamma^*), \tag{6.134}$$

which characterizes the support of the optimal transport plan. When the cost function is the squared Euclidean distance, $c(x, y) = \|x - y\|^2$, the transport problem reduces to finding a convex function $u(x)$ whose gradient defines the optimal transport map. This leads to the **Monge-Ampère equation**

$$\det D^2 u(x) = \frac{f(x)}{g(\nabla u(x))} \tag{6.135}$$

which provides a **geometric interpretation** of optimal transport: the transport map is given by the gradient of a convex function, ensuring volume preservation under mass transport. In this setting, the transport map satisfies

$$T(x) = \nabla u(x), \tag{6.136}$$

which is the Brenier map for the quadratic cost function. The induced transport cost is given by the **Wasserstein-2 distance**

$$W_2(\mu, \nu) = \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{X \times Y} \|x - y\|^2 d\gamma(x, y) \right)^{1/2}. \tag{6.137}$$

A fundamental computational approach to solving optimal transport problems is **entropic regularization**, which introduces an entropy term

$$\epsilon H(\gamma) = \epsilon \int_{X \times Y} \gamma(x, y) \log \gamma(x, y) dx dy, \tag{6.138}$$

leading to the **regularized optimal transport problem**

$$\inf_{\gamma \in \Pi(\mu,\nu)} \int_{X \times Y} c(x, y) d\gamma(x, y) + \epsilon H(\gamma). \tag{6.139}$$

These equations reveal deep connections between optimal transport, geometric analysis, and functional inequalities, providing a unifying framework for problems in probability, physics, and machine learning.

6.7.2 Fenchel-Rockafellar duality theorem

The **Fenchel-Rockafellar duality theorem** is a cornerstone result in convex analysis, establishing a deep connection between a primal convex optimization problem and its associated dual problem via the framework of convex conjugates and subdifferential calculus. It generalizes classical duality principles by employing functional analytic techniques and provides a powerful formulation for the optimality conditions in convex minimization problems. The theorem holds in the setting of a Banach space X paired with its continuous dual space X^*, and its implications extend to a broad range of optimization and variational problems.

Consider two proper, convex, and lower semicontinuous functions $f : X \to \mathbb{R} \cup \{+\infty\}$ and $g : X \to \mathbb{R} \cup \{+\infty\}$. The **primal optimization problem** is given by

$$\inf_{x \in X} \{f(x) + g(x)\}, \tag{6.140}$$

which seeks to minimize the sum of these two convex functions over the domain X. The associated **Fenchel conjugate function** (also referred to as the **Legendre-Fenchel transform**) is defined as

$$f^*(x^*) = \sup_{x \in X} \{\langle x^*, x\rangle - f(x)\}, \quad x^* \in X^*, \tag{6.141}$$

which captures the maximum possible affine lower bound of $f(x)$. Similarly, the conjugate of g is given by

$$g^*(x^*) = \sup_{x \in X} \{\langle x^*, x\rangle - g(x)\}, \quad x^* \in X^*. \tag{6.142}$$

The **Fenchel dual problem** is then formulated as

$$\sup_{x^* \in X^*} \{-(f^* + g^*)(x^*)\}. \tag{6.143}$$

The Fenchel-Rockafellar duality theorem states that under appropriate constraint qualifications, strong duality holds, meaning that the optimal values of the primal and dual problems coincide:

$$\inf_{x \in X} \{f(x) + g(x)\} = \sup_{x^* \in X^*} \{-(f^* + g^*)(x^*)\}. \tag{6.144}$$

To ensure strong duality, a sufficient condition is the existence of a point $x_0 \in X$ such that

$$x_0 \in \text{dom}(f) \cap \text{dom}(g), \quad \text{and} \quad 0 \in \text{core}(\text{dom}(g) - x_0), \tag{6.145}$$

where the **effective domain** of a function is defined as

$$\text{dom}(f) = \{x \in X \mid f(x) < +\infty\}, \tag{6.146}$$

and the **core** (or algebraic interior) of a set S is given by

$$\text{core}(S) = \{x \in S \mid \forall y \in X, \exists t > 0 \text{ such that } x + t(y - x) \in S\}. \tag{6.147}$$

This constraint qualification ensures that the subdifferential intersection necessary for optimality is nonempty. In particular, the subdifferential of a convex function f at a point x is defined as

$$\partial f(x) = \{x^* \in X^* \mid f(y) \geq f(x) + \langle x^*, y - x\rangle, \quad \forall y \in X\}. \tag{6.148}$$

For optimality, we require that the subdifferentials of f and g satisfy

$$0 \in \partial f(x) + \partial g(x), \tag{6.149}$$

which guarantees the existence of a Lagrange multiplier x^* such that

$$x^* \in \partial f(x), \quad -x^* \in \partial g(x). \tag{6.150}$$

That is, the dual optimizer x^* must belong to the intersection

$$\partial f(x) \cap (-\partial g(x)) \neq \emptyset. \tag{6.151}$$

This condition provides the necessary and sufficient optimality criterion in terms of subdifferential calculus. The theorem extends naturally to infinite-dimensional settings, where it serves as a foundation for convex variational problems in Hilbert and Banach spaces. By employing the Fenchel conjugates, it allows for a reformulation of many classical variational problems in dual terms, enabling powerful analytical techniques such as saddle-point theory, strong convexity conditions, and dual space representations. The theorem's implications in optimization, functional analysis, and mathematical economics highlight its fundamental role in modern analysis.

7 Open Set Learning

Open set learning is a foundational challenge in statistical learning theory and pattern recognition, characterized by the necessity of classifying data instances drawn from both known and unknown distributions. Consider a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where each instance $\mathbf{x}_i \in \mathbb{R}^d$ is associated with a label y_i from a finite set of known classes $\mathcal{Y}_{\text{train}}$. The fundamental challenge in open set learning arises when test samples are drawn from a distribution P_{test} that includes both the training distribution P_{train} and an additional unknown component P_{unknown}, such that

$$P_{\text{test}} = P_{\text{train}} + P_{\text{unknown}}, \quad P_{\text{unknown}} \cap P_{\text{train}} = \emptyset. \tag{7.1}$$

In classical closed-set classification, one seeks to find a function $f : \mathbb{R}^d \to \mathbb{R}^C$ that minimizes an empirical risk functional of the form

$$\mathcal{R}_{\text{emp}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\text{train}}}[\ell(f(\mathbf{x}), y)] \tag{7.2}$$

where $\ell : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ is a loss function such as categorical cross-entropy:

$$\ell(f(\mathbf{x}), y) = -\sum_{c=1}^{C} \nVdash[y = c] \log f_c(\mathbf{x}). \tag{7.3}$$

However, in the presence of unknown classes, minimizing only this risk function leads to overconfident misclassifications when a sample $\mathbf{x}_u \sim P_{\text{unknown}}$ is assigned to one of the known classes. This necessitates the introduction of a rejection mechanism, leading to an extended open set risk functional

$$\mathcal{R}_{\text{open}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\text{train}}}[\ell(f(\mathbf{x}), y)] + \lambda \mathbb{E}_{\mathbf{x}_u \sim P_{\text{unknown}}}[\ell_{\text{reject}}(f(\mathbf{x}_u))] \tag{7.4}$$

where ℓ_{reject} is a loss function that penalizes misclassification of unknown samples. A common approach is to introduce a threshold-based rejection criterion where the classification function f satisfies

$$\ell_{\text{reject}}(f(\mathbf{x}_u)) = \max_c f_c(\mathbf{x}_u) - \tau, \tag{7.5}$$

where τ is a rejection threshold such that if $\max_c f_c(\mathbf{x}_u) < \tau$, the sample is assigned to an unknown category. The introduction of this threshold formally defines an open space $\mathcal{O}(f)$, given by

$$\mathcal{O}(f) = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; \max_c f_c(\mathbf{x}) < \tau \right\}, \tag{7.6}$$

which is crucial in minimizing the open space risk introduced by Scheirer et al. (2013):

$$\mathcal{R}_{\text{open}}(f) = \int_{\mathcal{O}(f)} P(\mathbf{x}) d\mathbf{x}. \tag{7.7}$$

Incorporating open space risk minimization into the learning framework, the overall objective function becomes

$$\mathcal{L}(f) = \mathcal{R}_{\text{emp}}(f) + \beta \mathcal{R}_{\text{open}}(f), \tag{7.8}$$

where β is a hyperparameter that balances classification accuracy with rejection performance. The underlying theoretical framework necessitates consideration of both discriminative and generative approaches. In discriminative modeling, the classification function f can be regularized to enforce an explicit margin γ for open set rejection:

$$\min_{\mathbf{x}_u \sim P_{\text{unknown}}} \max_c f_c(\mathbf{x}_u) \leq \gamma. \tag{7.9}$$

In generative modeling, the problem is framed as estimating the likelihood $P(\mathbf{x} \mid y)$ under a known class distribution model, such as a Gaussian Mixture Model (GMM):

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{7.10}$$

where $\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$ represents a multivariate Gaussian distribution with mean μ_k and covariance matrix Σ_k, and π_k is the prior weight of the k-th component. A likelihood-based rejection rule is then applied:

$$P(\mathbf{x}) < \delta \Rightarrow \text{unknown}. \tag{7.11}$$

There are several class distribution models in which the open set learning problem can be framed as estimating the likelihood $P(\mathbf{x} \mid y)$. To state all those class distribution models we first define few quantities. Let $\mathcal{X} \subset \mathbb{R}^d$ be the input feature space and $\mathcal{Y} = \{1, 2, \ldots, C\}$ be the known set of class labels. We assume that samples are drawn from a joint probability distribution $P(\mathbf{x}, y)$, where the true distribution of unknown classes is not available during training. Open Set Learning requires estimating a function $f : \mathcal{X} \to \mathcal{Y} \cup \{\varnothing\}$, where $\varnothing$ represents an unknown class.

1. **Gaussian Distribution Model (GDM)** The fundamental assumption is that the feature distribution of each known class follows a **multivariate normal distribution** parameterized by the mean vector μ_c and covariance matrix Σ_c. The likelihood of a given sample $\mathbf{x}$ belonging to class c is:

$$P(\mathbf{x} \mid y = c) = \frac{1}{(2\pi)^{d/2} |\Sigma_c|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c) \right). \tag{7.12}$$

   To quantify the confidence in assigning $\mathbf{x}$ to class c, we compute the **Mahalanobis distance**:

$$D_M(\mathbf{x}, \mu_c, \Sigma_c) = \sqrt{(\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c)}. \tag{7.13}$$

   A sample is rejected as unknown if:

$$\min_c D_M(\mathbf{x}, \mu_c, \Sigma_c) > \delta, \tag{7.14}$$

   where δ is a threshold chosen based on extreme value statistics.

2. **Gaussian Mixture Model (GMM)** The Gaussian assumption can be generalized using a **Gaussian Mixture Model (GMM)**, which represents each class as a weighted sum of multiple Gaussian components:

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{7.15}$$

   where π_k are the mixture weights satisfying $\sum_{k=1}^{K} \pi_k = 1$, and each Gaussian component $\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$ is given by:

$$\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right). \tag{7.16}$$

   Unknown samples are rejected based on low **maximum likelihood estimation (MLE)** scores:

$$\max_c P(\mathbf{x} \mid y = c) < \tau. \tag{7.17}$$

   where τ is a predefined threshold.

3. **Dirichlet Process Gaussian Mixture Model (DP-GMM)** A **Dirichlet Process (DP)** prior can be introduced to allow the number of mixture components K to grow dynamically. The prior over the mixture weights follows:

$$\pi_k \sim \mathrm{Dir}(\alpha), \tag{7.18}$$

where α is the concentration parameter controlling cluster sparsity. This enables automatic adaptation of the number of mixture components to better capture class distributions. The likelihood of $\mathbf{x}$ follows:

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{7.19}$$

but with a nonparametric prior that ensures more flexible decision boundaries.

4. **Extreme Value Theory (EVT) Models** The **tail distribution** of softmax probabilities is modeled using an **Extreme Value Theorem (EVT)** approach. Given softmax scores $\sigma_c(\mathbf{x})$, we fit a **Weibull distribution** to the tail:

$$P_{\mathrm{Weibull}}(x) = 1 - \exp\left(-\left(\frac{x-\lambda}{\kappa}\right)^{\beta}\right). \tag{7.20}$$

A sample $\mathbf{x}$ is rejected if:

$$\max_c \sigma_c(\mathbf{x}) < \tau_{\mathrm{EVT}}. \tag{7.21}$$

5. **Bayesian Neural Networks (BNNs)** BNNs introduce uncertainty estimation by placing **priors** over network weights:

$$w \sim P(w). \tag{7.22}$$

Posterior inference is performed via **Bayesian updating**:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, w) P(w \mid \mathcal{D}) dw. \tag{7.23}$$

A sample is rejected if the **entropy** of the predictive distribution is high:

$$H[P(y \mid \mathbf{x})] = -\sum_c P(y = c \mid \mathbf{x}) \log P(y = c \mid \mathbf{x}). \tag{7.24}$$

6. **Support Vector Models (OC-SVM and SVDD) One-Class SVM (OC-SVM):** Finds a separating hyperplane $w^T \phi(\mathbf{x}) + b = 0$ such that:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{7.25}$$

subject to $w^T \phi(\mathbf{x}) + b \geq \rho$. A sample is rejected if:

$$w^T \phi(\mathbf{x}) + b < \rho. \tag{7.26}$$

7. **Support Vector Data Description (SVDD):** Finds a **minimum enclosing hypersphere** with center $\mathbf{c}$ and radius R:

$$\min_{R,\mathbf{c}} R^2 + C \sum_i \xi_i \tag{7.27}$$

subject to:

$$\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2 + \xi_i. \tag{7.28}$$

A sample is rejected if:

$$\|\phi(\mathbf{x}) - \mathbf{c}\| > R. \tag{7.29}$$

Each model presents a trade-off between expressivity, computational complexity, and interpretability. The choice of model for Open Set Learning depends on the underlying data distribution and the required rejection confidence.

A distance-based approach further refines the rejection mechanism by computing the Mahalanobis distance

$$D_M(\mathbf{x}, \mu_c, \Sigma_c) = \sqrt{(\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c)} \tag{7.30}$$

and rejecting samples that satisfy

$$\min_c D_M(\mathbf{x}, \mu_c, \Sigma_c) > \delta. \tag{7.31}$$

An alternative Bayesian formulation considers the posterior probability of a sample belonging to a known class:

$$P(y \in \mathcal{Y}_{\text{train}} \mid \mathbf{x}) = \frac{\sum_{c=1}^{C} P(\mathbf{x} \mid y = c) P(y = c)}{P(\mathbf{x})}. \tag{7.32}$$

Defining a prior probability $P(y^* \notin \mathcal{Y}_{\text{train}}) = \alpha$, a Bayesian rejection model classifies a sample as unknown if

$$P(y^* \notin \mathcal{Y}_{\text{train}} \mid \mathbf{x}) = 1 - P(y \in \mathcal{Y}_{\text{train}} \mid \mathbf{x}) > \alpha. \tag{7.33}$$

By leveraging a combination of margin constraints, likelihood estimation, and probabilistic modeling, open set learning provides a rigorous mathematical framework for handling unknown distributions while maintaining classification performance on known data. The continued development of theoretical models and practical algorithms in this domain is essential for deploying reliable machine learning systems in real-world scenarios where distributional shifts and novel categories are unavoidable.

7.1 Mahalanobis Distance

The Mahalanobis distance is a fundamental metric in Open Set Learning (OSL), where the primary objective is to correctly classify known samples while rigorously rejecting unknown samples. Given a feature space $\mathbb{R}^d$, let the data distribution of known classes be characterized by a mean feature vector and a covariance structure. The Mahalanobis distance is an extension of the Euclidean distance that takes into account correlations between different feature dimensions and properly normalizes for variance in different directions of the feature space. If $\mathbf{x} \in \mathbb{R}^d$ is a given feature vector extracted from a trained deep neural network and $\boldsymbol{\mu}_c \in \mathbb{R}^d$ represents the mean feature vector of a particular class c, then the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ describes the spread of features around their respective class means. The Mahalanobis distance of the feature vector $\mathbf{x}$ from the mean representation of class c is given by

$$D_M(\mathbf{x}, c) = \sqrt{(\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_c)}. \tag{7.34}$$

This metric effectively measures how many standard deviations away the point $\mathbf{x}$ is from the mean $\boldsymbol{\mu}_c$ when considering the full covariance structure. The inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$ accounts for correlations between different feature dimensions, ensuring that directions of high variance contribute less to the distance measurement, while directions of low variance contribute more. If the covariance matrix $\boldsymbol{\Sigma}$ is diagonal, the Mahalanobis distance reduces to a normalized Euclidean distance, with each feature being scaled by the inverse of its standard deviation. Explicitly, if

$$\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2), \tag{7.35}$$

then

$$D_M(\mathbf{x}, c) = \sqrt{\sum_{i=1}^{d} \frac{(x_i - \mu_{c,i})^2}{\sigma_i^2}}. \tag{7.36}$$

However, in high-dimensional feature spaces where correlations exist between different feature components, the full covariance matrix must be considered. The primary utility of Mahalanobis distance in Open Set Learning stems from its ability to identify samples that deviate significantly from the known class distributions. The decision rule for classification in Open Set Learning using Mahalanobis distance is defined as follows: for a given test sample $\mathbf{x}$, the predicted label $\hat{y}$ is assigned based on

$$\hat{y} = \arg\min_{c \in \mathcal{K}} D_M(\mathbf{x}, c), \tag{7.37}$$

where $\mathcal{K}$ is the set of known classes. However, in Open Set Learning, the classification decision must also incorporate a rejection criterion for unknown samples. This is accomplished by introducing a rejection threshold τ, such that if

$$\min_{c \in \mathcal{K}} D_M(\mathbf{x}, c) > \tau, \tag{7.38}$$

then $\mathbf{x}$ is rejected as an unknown sample. The threshold τ is typically determined based on the distribution of Mahalanobis distances computed from the training set. Specifically, if the Mahalanobis distances of training samples follow a Gaussian distribution

$$D_M(\mathbf{x}, c) \sim \mathcal{N}(\mu_D, \sigma_D^2), \tag{7.39}$$

then the threshold τ can be set as

$$\tau = \mu_D + \lambda \sigma_D, \tag{7.40}$$

where λ is a hyperparameter controlling the rejection sensitivity. A larger value of λ increases the likelihood of rejecting unknown samples but may also lead to the rejection of some in-distribution samples. Conversely, a smaller λ results in fewer rejections, potentially increasing the misclassification of unknown samples as known ones. In high-dimensional feature spaces, the covariance matrix $\mathbf{\Sigma}$ may become singular or ill-conditioned, making the direct computation of $\mathbf{\Sigma}^{-1}$ unstable. To address this, a regularized covariance estimate is often employed:

$$\mathbf{\Sigma}_\lambda = \mathbf{\Sigma} + \lambda \mathbf{I}, \tag{7.41}$$

where $\lambda > 0$ is a small regularization parameter, ensuring numerical stability. The corresponding regularized Mahalanobis distance is then given by

$$D_M^\lambda(\mathbf{x}, c) = \sqrt{(\mathbf{x} - \boldsymbol{\mu}_c)^\top \mathbf{\Sigma}_\lambda^{-1} (\mathbf{x} - \boldsymbol{\mu}_c)}. \tag{7.42}$$

This formulation ensures robust covariance estimation and improves the generalization ability of the Mahalanobis distance metric. In deep Open Set Learning settings, feature representations obtained from neural networks may not be perfectly Gaussian, necessitating additional adaptation techniques. One approach is to learn feature embeddings such that Mahalanobis distance becomes more effective for Open Set Recognition. If $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$ is a neural network feature extractor parameterized by θ, then the loss function can be augmented to explicitly minimize intra-class Mahalanobis distances while maximizing inter-class separability:

$$\mathcal{L}_M = \sum_{(x,y) \in \mathcal{D}_{\text{train}}} D_M(f_\theta(\mathbf{x}), y). \tag{7.43}$$

Furthermore, in order to improve the rejection of out-of-distribution samples, an auxiliary dataset $\mathcal{D}_{\text{OOD}}$ containing unknown samples can be used. The out-of-distribution loss is given by

$$\mathcal{L}_{\text{OOD}} = \sum_{\mathbf{x} \in \mathcal{D}_{\text{OOD}}} \max(0, \tau - \min_{c \in \mathcal{K}} D_M(f_\theta(\mathbf{x}), c)). \tag{7.44}$$

This enforces a constraint that encourages the Mahalanobis distance of unknown samples to remain above the rejection threshold τ, improving the ability of the model to detect unknown samples. The combined objective function used in Open Set Learning can then be formulated as

$$\mathcal{L} = \mathcal{L}_M + \lambda_{\text{OOD}} \mathcal{L}_{\text{OOD}}, \tag{7.45}$$

where λ_{OOD} is a balancing hyperparameter. In addition to the rejection-based approach, Mahalanobis distance can be integrated with probabilistic modeling techniques such as Gaussian mixture models (GMMs) for better open set decision boundaries. Given a mixture model with C Gaussian components, the probability of a feature vector $\mathbf{x}$ belonging to class c is computed as

$$P(c \mid \mathbf{x}) = \frac{\pi_c N(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{c' \in \mathcal{K}} \pi_{c'} N(\mathbf{x} \mid \boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'})}, \tag{7.46}$$

where π_c is the prior probability of class c and

$$N(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_c|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) \right) \tag{7.47}$$

is the Gaussian density function. This formulation enables a soft probabilistic decision rule for Open Set Learning. The Mahalanobis distance, due to its statistical foundation, provides a rigorous means of identifying unknown samples in Open Set Learning by leveraging class-specific feature distributions. Its ability to incorporate covariance structures makes it significantly more robust than Euclidean distance, leading to improved open set classification performance in high-dimensional spaces.

7.2 Bayesian Formulation in Open Set Learning

Open Set Learning (OSL) aims to handle scenarios where the model encounters unseen classes during inference. A Bayesian approach provides a principled probabilistic framework for quantifying uncertainty and distinguishing known from unknown instances. This involves incorporating prior knowledge, likelihood estimation, and posterior inference to make robust predictions under open-set conditions.

We have to use Bayesian Decision Theory in Open Set Learning. In the Bayesian framework, given an input $\mathbf{x} \in \mathbb{R}^d$, the goal is to estimate the posterior probability of class membership:

$$P(y \mid \mathbf{x}, \mathcal{D}) \tag{7.48}$$

where y belongs to the set of known classes $\mathcal{Y}_{\text{known}}$, and $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ represents the training data. The Bayesian formulation considers a prior distribution $P(y)$ over classes and updates it using the likelihood $P(\mathbf{x} \mid y)$ via Bayes' theorem:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \frac{P(\mathbf{x} \mid y) P(y)}{P(\mathbf{x})} \tag{7.49}$$

where the evidence term is given by marginalization over all known classes:

$$P(\mathbf{x}) = \sum_{y' \in \mathcal{Y}_{\text{known}}} P(\mathbf{x} \mid y') P(y'). \tag{7.50}$$

For open-set scenarios, an additional unknown class y_u is introduced, leading to:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \frac{P(\mathbf{x} \mid y) P(y)}{P(\mathbf{x}) + P(\mathbf{x} \mid y_u) P(y_u)} \tag{7.51}$$

where $P(y_u)$ is the prior probability of encountering an unknown class. If $P(y_u)$ is too low, the model may overcommit to known classes. We have to model the Likelihood via Generative Distributions. To estimate $P(\mathbf{x} \mid y)$, a Bayesian model assumes a parametric likelihood such as a Gaussian mixture:

$$P(\mathbf{x} \mid y = k) = \sum_{j=1}^{M_k} \pi_j^{(k)} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j^{(k)}, \boldsymbol{\Sigma}_j^{(k)}). \tag{7.52}$$

The parameters $\boldsymbol{\theta} = \{\pi_j^{(k)}, \boldsymbol{\mu}_j^{(k)}, \boldsymbol{\Sigma}_j^{(k)}\}$ are inferred using Maximum A Posteriori (MAP) estimation:

$$P(\boldsymbol{\theta} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \boldsymbol{\theta}) P(\boldsymbol{\theta}). \tag{7.53}$$

By integrating out uncertainty in parameters, a Bayesian predictive distribution is obtained:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}. \tag{7.54}$$

For open-set detection, if the posterior entropy

$$H(y \mid \mathbf{x}, \mathcal{D}) = -\sum_{y \in \mathcal{Y}_{\text{known}}} P(y \mid \mathbf{x}, \mathcal{D}) \log P(y \mid \mathbf{x}, \mathcal{D}) \tag{7.55}$$

exceeds a threshold, the sample is classified as unknown. Bayesian Neural Networks can also be used for Open Set Learning. A Bayesian neural network (BNN) introduces a prior distribution over the network parameters $\boldsymbol{w}$, leading to a posterior distribution:

$$P(\boldsymbol{w} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \boldsymbol{w}) P(\boldsymbol{w})}{P(\mathcal{D})}. \tag{7.56}$$

The predictive posterior for a test input $\mathbf{x}$ is then given by:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, \boldsymbol{w}) P(\boldsymbol{w} \mid \mathcal{D}) d\boldsymbol{w}. \tag{7.57}$$

Since this integral is intractable, approximation techniques such as Variational Inference (VI) or Monte Carlo Dropout are used. In VI, the posterior is approximated by a parametric distribution $q_\phi(\boldsymbol{w})$:

$$\min_\phi D_{\text{KL}}(q_\phi(\boldsymbol{w}) \parallel P(\boldsymbol{w} \mid \mathcal{D})) \tag{7.58}$$

which results in an evidence lower bound (ELBO):

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\boldsymbol{w})} [\log P(\mathcal{D} \mid \boldsymbol{w})] - D_{\text{KL}}(q_\phi(\boldsymbol{w}) \parallel P(\boldsymbol{w})). \tag{7.59}$$

Predictions are made using multiple stochastic forward passes, yielding an uncertainty estimate through variance:

$$\mathbb{V}[P(y \mid \mathbf{x}, \mathcal{D})]. \tag{7.60}$$

High variance corresponds to greater epistemic uncertainty, signaling an open-set sample. There are several Bayesian Nonparametric Methods for Open Set Learning. Bayesian nonparametric models, such as Gaussian Processes (GPs), provide an alternative approach. A GP places a prior over functions:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{7.61}$$

The posterior mean and variance for a new point $\mathbf{x}_*$ given training data $\mathcal{D}$ are:

$$\mu_* = k_*^\top (K + \sigma^2 I)^{-1} \mathbf{y} \tag{7.62}$$

$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^\top (K + \sigma^2 I)^{-1} k_*. \tag{7.63}$$

If σ_*^2 is large, the sample is flagged as unknown. Let's now discuss the Bayesian Open Set Recognition with Dirichlet Distributions. A Bayesian framework can incorporate Dirichlet distributions for modeling class probabilities:

$$P(\mathbf{p} \mid \boldsymbol{\alpha}) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_{i=1}^{K} p_i^{\alpha_i - 1}. \tag{7.64}$$

For open-set detection, an unknown class component is introduced, leading to an augmented Dirichlet prior:

$$P(\mathbf{p} \mid \boldsymbol{\alpha}, \alpha_u) \propto \prod_{i=1}^{K} p_i^{\alpha_i - 1} \cdot (1 - \sum_{i=1}^{K} p_i)^{\alpha_u - 1}. \tag{7.65}$$

If the posterior probability mass assigned to the unknown component is significant, the sample is rejected as open-set.

In conclusion, A Bayesian approach to Open Set Learning provides a principled uncertainty quantification mechanism, distinguishing unknown from known samples by leveraging posterior probabilities, entropy-based rejection criteria, and epistemic uncertainty estimates. The use of Bayesian Neural Networks, Gaussian Processes, and Dirichlet distributions enables robust open-set recognition, improving model reliability in real-world applications.

7.3 Gaussian Mixture Model (GMM)

In generative modeling, the open set learning problem is fundamentally rooted in the estimation of the likelihood function $P(x \mid y)$, where $x \in \mathbb{R}^d$ represents an observed data sample in a d-dimensional feature space, and y is a discrete class label. The Gaussian Mixture Model (GMM) is an ideal candidate for modeling this likelihood function due to its ability to approximate arbitrary density functions as a superposition of Gaussian components. Formally, the GMM models the conditional density function $P(x \mid y)$ as a weighted sum of multivariate normal distributions:

$$P(x \mid y) = \sum_{k=1}^{K} \pi_k^{(y)} \mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}). \tag{7.66}$$

where each Gaussian component k for class y is parameterized by a mean vector $\mu_k^{(y)} \in \mathbb{R}^d$, a covariance matrix $\Sigma_k^{(y)} \in \mathbb{R}^{d \times d}$, and a mixture weight $\pi_k^{(y)}$ satisfying the normalization constraint:

$$\sum_{k=1}^{K} \pi_k^{(y)} = 1, \quad 0 \leq \pi_k^{(y)} \leq 1. \tag{7.67}$$

The probability density function of each multivariate normal component is given explicitly by:

$$\mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}) = \frac{1}{(2\pi)^{d/2} |\Sigma_k^{(y)}|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu_k^{(y)})^T \Sigma_k^{(y)^{-1}} (x - \mu_k^{(y)}) \right). \tag{7.68}$$

By expanding the quadratic form in the exponent, the probability density function can be rewritten in terms of Mahalanobis distance:

$$\mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}) = \frac{1}{(2\pi)^{d/2} |\Sigma_k^{(y)}|^{1/2}} \exp\left(-\frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (x_i - \mu_{ki}^{(y)}) \Sigma_{k,ij}^{(y)^{-1}} (x_j - \mu_{kj}^{(y)}) \right). \tag{7.69}$$

The total log-likelihood of the data under the GMM model is given by:

$$\log P(x \mid y) = \log \sum_{k=1}^{K} \pi_k^{(y)} \mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}). \tag{7.70}$$

Direct computation of the log-sum can lead to numerical instability, which is mitigated using the **log-sum-exp trick**:

$$\log P(x \mid y) = m + \log \sum_{k=1}^{K} \pi_k^{(y)} \exp\left(-\frac{1}{2} (x - \mu_k^{(y)})^T \Sigma_k^{(y)^{-1}} (x - \mu_k^{(y)}) - m \right), \tag{7.71}$$

where

$$m = \max_k \left\{ -\frac{1}{2}(x - \mu_k^{(y)})^T \Sigma_k^{(y)^{-1}} (x - \mu_k^{(y)}) \right\} \tag{7.72}$$

is chosen to stabilize exponentiation. For open set recognition, an input sample x is classified as an out-of-distribution (OOD) sample if its likelihood $P(x \mid y)$ falls below a predefined threshold τ, i.e.,

$$\log P(x \mid y) < \tau \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{7.73}$$

An alternative formulation leverages Bayesian inference to determine the most probable class y^* given an observed sample x. Using Bayes' theorem,

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{\sum_{y'} P(x \mid y')P(y')}, \tag{7.74}$$

where $P(y)$ is the prior probability of class y, which is often assumed uniform over the set of known classes. The most probable class is then determined via the maximum a posteriori (MAP) estimation:

$$y^* = \arg\max_y P(y \mid x). \tag{7.75}$$

If $P(y^* \mid x)$ is below a rejection threshold η, the sample is assigned to an unknown category:

$$P(y^* \mid x) < \eta \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{7.76}$$

The training of the Gaussian Mixture Model parameters $\{\mu_k^{(y)}, \Sigma_k^{(y)}, \pi_k^{(y)}\}$ is performed using the Expectation-Maximization (EM) algorithm. Given a dataset $\{x_1, x_2, \dots, x_N\}$, the **E-step** computes the posterior responsibility of each Gaussian component:

$$r_k^{(y)}(x_i) = \frac{\pi_k^{(y)} \mathcal{N}(x_i \mid \mu_k^{(y)}, \Sigma_k^{(y)})}{\sum_{j=1}^K \pi_j^{(y)} \mathcal{N}(x_i \mid \mu_j^{(y)}, \Sigma_j^{(y)})}. \tag{7.77}$$

In the **M-step**, the parameters are updated as follows:

$$\mu_k^{(y)} = \frac{\sum_{i=1}^N r_k^{(y)}(x_i) x_i}{\sum_{i=1}^N r_k^{(y)}(x_i)} \tag{7.78}$$

$$\Sigma_k^{(y)} = \frac{\sum_{i=1}^N r_k^{(y)}(x_i)(x_i - \mu_k^{(y)})(x_i - \mu_k^{(y)})^T}{\sum_{i=1}^N r_k^{(y)}(x_i)} \tag{7.79}$$

$$\pi_k^{(y)} = \frac{1}{N} \sum_{i=1}^N r_k^{(y)}(x_i). \tag{7.80}$$

An alternative approach to OOD detection is based on the **Mahalanobis distance**, which measures the squared deviation of a sample from a Gaussian component:

$$D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)}) = (x - \mu_k^{(y)})^T \Sigma_k^{(y)^{-1}} (x - \mu_k^{(y)}). \tag{7.81}$$

If $D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)})$ exceeds a predefined threshold γ, then the sample is classified as OOD:

$$D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)}) > \gamma \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{7.82}$$

The Gaussian Mixture Model provides a theoretically rigorous and computationally efficient approach for open set learning, leveraging probabilistic likelihood estimation, Bayesian inference, and robust distance measures to distinguish in-distribution and out-of-distribution samples. The parameter optimization via Expectation-Maximization ensures maximum likelihood estimation of the mixture components, and the decision criteria based on likelihood thresholds or Mahalanobis distances provide strong theoretical guarantees for identifying unknown samples.

7.4 Dirichlet Process Gaussian Mixture Model (DP-GMM)

In generative modeling, the open-set learning problem is fundamentally framed as estimating the conditional likelihood $P(x|y)$ under a **Dirichlet Process Gaussian Mixture Model (DP-GMM)**. The central challenge in open-set learning is to correctly handle data points that may arise from an **unknown category**, meaning that their underlying distribution differs from that of the training data. To achieve this, we employ a nonparametric Bayesian approach that allows for an **unbounded** number of mixture components, ensuring that the model can flexibly introduce new clusters for previously unseen samples. This is accomplished by imposing a **Dirichlet Process (DP) prior** on the mixture component weights, which results in a countably infinite Gaussian Mixture Model (GMM), effectively allowing the number of clusters to grow dynamically with the data. Given an observed dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where each $x_i \in \mathbb{R}^d$ represents a sample in a d-dimensional feature space and y_i is its corresponding class label, the likelihood estimation problem can be expressed in the form

$$P(x|y) = \sum_{k=1}^{\infty} P(x|z=k, y) P(z=k|y) \tag{7.83}$$

where z represents a latent variable denoting the cluster assignment. The likelihood $P(x|z = k, y)$ follows a multivariate Gaussian distribution parameterized by cluster-specific mean μ_k and covariance Σ_k, given by

$$P(x|z=k, y) = \mathcal{N}(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right). \tag{7.84}$$

Since the number of mixture components is unknown, a **Dirichlet Process (DP) prior** is imposed on the cluster assignment probabilities, leading to the hierarchical model

$$G \sim \mathrm{DP}(\alpha, G_0), \quad \theta_k \sim G, \quad x_i|z_i, \theta_{z_i} \sim P(x|\theta_{z_i}), \tag{7.85}$$

where G is a discrete random probability measure drawn from a Dirichlet Process with **concentration parameter** $\alpha > 0$ and base distribution G_0, which defines the prior over component parameters. The construction of the DP-GMM is commonly represented using the **stick-breaking process**, where the mixture weights are recursively defined as

$$\pi_k = v_k \prod_{j=1}^{k-1} (1 - v_j), \quad v_k \sim \mathrm{Beta}(1, \alpha) \tag{7.86}$$

ensuring the normalization constraint

$$\sum_{k=1}^{\infty} \pi_k = 1. \tag{7.87}$$

Given this formulation, the posterior distribution over cluster assignments follows

$$P(z=k|x,y) = \frac{P(x|z=k,y)P(z=k|y)}{\sum_{j=1}^{\infty} P(x|z=j,y)P(z=j|y)}. \tag{7.88}$$

The open-set learning problem requires the ability to **identify and appropriately model unseen categories**. This is naturally handled by the **Chinese Restaurant Process (CRP) representation** of the DP, where a new data point x^* is assigned to a **new cluster** with probability

$$P(z^* = K+1|x^*, y^*) = \frac{\alpha}{\alpha + N} \tag{7.89}$$

where N is the total number of previously observed samples. Consequently, the likelihood of an unseen data point under the DP-GMM model expands as

$$P(x^*|y^*) = \sum_{k=1}^{K} P(x^*|z^* = k, y^*)P(z^* = k|y^*) + P(x^*|z^* = K+1, y^*)P(z^* = K+1|y^*). \tag{7.90}$$

A sample is **classified as OOD** (out-of-distribution) if its **posterior likelihood** falls below a learned threshold τ, which can be determined using a Bayesian uncertainty criterion, leading to the decision rule

$$\text{if } P(x|y) < \tau, \quad \text{then } x \text{ is classified as OOD.} \tag{7.91}$$

To further quantify the **uncertainty in cluster assignments**, the **Shannon entropy** of the posterior is computed as

$$H(x) = -\sum_{k=1}^{K+1} P(z = k|x, y) \log P(z = k|x, y). \tag{7.92}$$

A high entropy indicates significant **uncertainty**, suggesting that the sample does not belong to any of the known categories and should be treated as an **open-set** example. If entropy exceeds a critical threshold, the sample is **automatically assigned to a new cluster**, thereby allowing the model to flexibly accommodate previously unseen classes. The hierarchical Bayesian formulation of the DP-GMM further provides a **robust probabilistic framework** for managing **inherent ambiguity** in classification by leveraging both the **prior knowledge encoded in** G_0 and the **adaptive nature of the Dirichlet Process**, which dynamically introduces new components as necessary. From an inference perspective, model parameters (μ_k, Σ_k) and mixture proportions π_k are estimated using **variational inference** or **Gibbs sampling**, where the posterior updates involve integrating over all possible assignments z given the prior over cluster structure. The posterior distribution over mixture components follows

$$P(\mu_k, \Sigma_k|\mathcal{D}) \propto P(\mathcal{D}|\mu_k, \Sigma_k)P(\mu_k, \Sigma_k), \tag{7.93}$$

where the prior $P(\mu_k, \Sigma_k)$ is typically chosen as a **conjugate Normal-Inverse-Wishart (NIW) distribution**, ensuring closed-form updates. The resulting Bayesian inference scheme ensures that uncertainty is **explicitly quantified** at every level of the learning process, making DP-GMM a **principled generative model for open-set learning**.

7.5 Conjugate Normal-Inverse-Wishart (NIW) distribution

In the context of generative modeling, the open set learning problem is rigorously framed as the estimation of the likelihood $P(x|y)$ for a data point x given a class label y, where y may belong to either a known class or an unknown class. This problem is addressed by modeling the likelihood $P(x|y)$ under a conjugate Normal-Inverse-Wishart (NIW) distribution, which provides a mathematically tractable and statistically rigorous framework for parameter estimation. The NIW distribution serves as a conjugate prior for the multivariate normal distribution with unknown mean μ and covariance Σ, enabling closed-form updates of the posterior distribution given observed data. The NIW distribution is parameterized by hyperparameters μ_0, λ, Ψ, and ν, where $\mu_0 \in \mathbb{R}^d$ is the prior mean, $\lambda > 0$ is a scaling factor for the prior mean, $\Psi \in \mathbb{R}^{d\times d}$ is a symmetric positive definite scale matrix, and $\nu > d-1$ is the degrees of freedom. The joint probability density function of the NIW distribution is given by:

$$P(\mu, \Sigma|\mu_0, \lambda, \Psi, \nu) = \text{NIW}(\mu, \Sigma|\mu_0, \lambda, \Psi, \nu) = \mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma) \cdot \mathcal{W}^{-1}(\Sigma|\Psi, \nu), \tag{7.94}$$

where $\mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma)$ is the multivariate normal distribution and $\mathcal{W}^{-1}(\Sigma|\Psi, \nu)$ is the inverse-Wishart distribution. The multivariate normal distribution is defined as:

$$\mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma) = \frac{1}{(2\pi)^{d/2}|\frac{1}{\lambda}\Sigma|^{1/2}} \exp\left(-\frac{\lambda}{2}(\mu - \mu_0)^T\Sigma^{-1}(\mu - \mu_0)\right), \tag{7.95}$$

and the inverse-Wishart distribution is defined as:

$$\mathcal{W}^{-1}(\Sigma|\Psi, \nu) = \frac{|\Psi|^{\nu/2}}{2^{\nu d/2}\Gamma_d\left(\frac{\nu}{2}\right)}|\Sigma|^{-(\nu+d+1)/2} \exp\left(-\frac{1}{2}\mathrm{tr}(\Psi\Sigma^{-1})\right), \tag{7.96}$$

where Γ_d is the multivariate gamma function, d is the dimensionality of the data, and tr denotes the trace of a matrix. Given a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, \ldots, K\}$ for known classes or $y_i = K + 1$ for an unknown class, the likelihood $P(x|y)$ is modeled as a multivariate normal distribution with class-specific parameters μ_y and Σ_y. These parameters are drawn from the NIW prior distribution:

$$P(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu) = \mathrm{NIW}(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu). \tag{7.97}$$

The posterior distribution of μ_y and Σ_y given the observed data $\mathcal{D}_y = \{x_i : y_i = y\}$ is also an NIW distribution, with updated hyperparameters $\mu_y^*, \lambda^*, \Psi^*, \nu^*$. The updated hyperparameters are derived as:

$$\mu_y^* = \frac{\lambda\mu_0 + N_y\bar{x}_y}{\lambda + N_y}, \tag{7.98}$$

$$\lambda^* = \lambda + N_y, \tag{7.99}$$

$$\Psi^* = \Psi + S_y + \frac{\lambda N_y}{\lambda + N_y}(\bar{x}_y - \mu_0)(\bar{x}_y - \mu_0)^T, \tag{7.100}$$

$$\nu^* = \nu + N_y, \tag{7.101}$$

where $N_y = |\mathcal{D}_y|$ is the number of data points in class y, $\bar{x}_y = \frac{1}{N_y}\sum_{x_i \in \mathcal{D}_y} x_i$ is the sample mean of the data points in class y, and $S_y = \sum_{x_i \in \mathcal{D}_y}(x_i - \bar{x}_y)(x_i - \bar{x}_y)^T$ is the scatter matrix for class y. The likelihood $P(x|y)$ is obtained by marginalizing over the parameters μ_y and Σ_y:

$$P(x|y) = \int\int P(x|\mu_y, \Sigma_y)P(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu)\, d\mu_y\, d\Sigma_y. \tag{7.102}$$

Due to the conjugacy of the NIW prior, this integral can be evaluated analytically, resulting in a multivariate Student's t-distribution:

$$P(x|y) = \mathcal{T}(x|\mu_y^*, \frac{\Psi^*}{\lambda^*(\nu^* - d + 1)}, \nu^* - d + 1), \tag{7.103}$$

where $\mathcal{T}(x|\mu, \Sigma, \nu)$ is the multivariate Student's t-distribution with mean μ, scale matrix Σ, and degrees of freedom ν. The multivariate Student's t-distribution is given by:

$$\mathcal{T}(x|\mu, \Sigma, \nu) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)(\nu\pi)^{d/2}|\Sigma|^{1/2}}\left(1 + \frac{1}{\nu}(x - \mu)^T\Sigma^{-1}(x - \mu)\right)^{-(\nu+d)/2}. \tag{7.104}$$

In the open set learning framework, the likelihood $P(x|y)$ is used to compute the probability that a new data point x belongs to a known class y. If $P(x|y)$ falls below a predefined threshold for all known classes y, the data point x is classified as belonging to an unknown class. This approach leverages the statistical properties of the NIW distribution and the multivariate Student's t-distribution to provide a rigorous and principled solution to the open set learning problem. The mathematical formulation ensures that the model can effectively distinguish between known and unknown classes while maintaining robustness and interpretability.

7.6 Extreme Value Theory (EVT) Models

In generative modeling, the open set learning problem concerns the ability of a model to distinguish between seen and unseen data by estimating the likelihood $P(\mathbf{x}|y)$. This problem is fundamentally linked to extreme value theory (EVT), which characterizes the statistical behavior of the tails of probability distributions. EVT provides a rigorous mathematical framework for modeling the behavior of rare events, which is crucial for identifying out-of-distribution (OOD) samples. Given an observed data point $\mathbf{x}$ and a class label y, the likelihood function $P(\mathbf{x}|y)$ exhibits tail properties that can be asymptotically approximated using EVT. This follows from the Fisher-Tippett-Gnedenko theorem, which states that the maximum of independent and identically distributed (i.i.d.) random variables, properly normalized, converges to one of three types of extreme value distributions: Gumbel, Fréchet, or Weibull. The key observation in open set recognition is that samples from an unknown class exhibit extreme deviations in their likelihood estimates, which can be rigorously modeled through EVT-based tail fitting.

The first step in this formulation is to define a log-likelihood function, which quantifies the probability of observing a sample given a particular class. Let the log-likelihood function be given by

$$L(\mathbf{x}) = \log P(\mathbf{x}|y) \tag{7.105}$$

where $L(\mathbf{x})$ captures the statistical relationship between the observed data and the conditional density function. In EVT-based approaches, one considers the distribution of extreme values of $L(\mathbf{x})$, specifically in the lower tail, since OOD samples tend to have significantly lower likelihoods. The fundamental result from EVT states that for a sufficiently large number of independent observations, the limiting distribution of the minimum values follows a Generalized Extreme Value (GEV) distribution, given by

$$F(z;\xi,\mu,\sigma) = \begin{cases} \exp\left(-\left(1+\frac{\xi(z-\mu)}{\sigma}\right)^{-1/\xi}\right), & \xi \neq 0 \\ \exp\left(-e^{-(z-\mu)/\sigma}\right), & \xi = 0 \end{cases} \tag{7.106}$$

where ξ is the shape parameter, μ is the location parameter, and σ is the scale parameter. In the context of likelihood estimation, the limiting behavior of $L(\mathbf{x})$ follows a Generalized Pareto Distribution (GPD), which describes the excess over a given threshold. The probability density function (PDF) of the GPD is given by

$$g(z;\xi,\beta) = \begin{cases} \frac{1}{\beta}\left(1+\frac{\xi z}{\beta}\right)^{-1-1/\xi}, & \xi \neq 0, z > 0 \\ \frac{1}{\beta}e^{-z/\beta}, & \xi = 0, z > 0 \end{cases} \tag{7.107}$$

where $z = L_{\max} - L(\mathbf{x})$ represents the deviation from the maximum likelihood observed in training data. The fundamental insight from EVT is that for sufficiently high thresholds, the distribution of these deviations converges to the GPD. This allows us to estimate the probability that a given likelihood falls below a certain threshold τ using

$$P(L(\mathbf{x}) < \tau) = 1 - \left(1 + \frac{\xi(\tau-\mu)}{\sigma}\right)^{-1/\xi} \tag{7.108}$$

which provides a principled way of defining OOD detection thresholds. A sample $\mathbf{x}$ is considered to be OOD if

$$P(\mathbf{x}|y) < \tau \tag{7.109}$$

where τ is selected based on the EVT-derived confidence interval. Given a set of training samples $\{\mathbf{x}_i\}_{i=1}^N$ from a known class, one can fit the EVT parameters by solving the maximum likelihood estimation (MLE) problem

$$\hat{\xi}, \hat{\sigma} = \arg\max_{\xi,\sigma} \sum_{i=1}^{N} \log g(L_{\max} - L(\mathbf{x}_i); \xi, \sigma). \tag{7.110}$$

This provides an optimal estimate of the tail parameters, ensuring that the EVT model accurately represents the behavior of extreme likelihood values. Once the parameters are estimated, one can compute a dynamic threshold for rejecting OOD samples by solving

$$\tau = \mu + \frac{\sigma}{\xi}\left(\left(\frac{1-p}{N}\right)^{-\xi} - 1\right) \tag{7.111}$$

where p is the desired false positive rate for in-distribution samples. The EVT-based formulation allows for a probabilistic characterization of whether a given sample belongs to a known or unknown class, rather than relying on heuristics. To incorporate this within a Bayesian framework, the posterior probability of a class given an observation is computed using Bayes' theorem as

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \tag{7.112}$$

Since EVT provides a distributional form for $P(\mathbf{x})$ under extreme value behavior, one can estimate the denominator using the cumulative probability function of the GPD:

$$P(\mathbf{x}) = \sum_{y'} P(\mathbf{x}|y')P(y'). \tag{7.113}$$

Replacing $P(\mathbf{x}|y)$ with the EVT-derived approximation, the posterior can be rewritten as

$$P(y|\mathbf{x}) = \frac{\left(1 + \frac{\xi(L(\mathbf{x})-\mu)}{\sigma}\right)^{-1/\xi} P(y)}{\sum_{y'} \left(1 + \frac{\xi(L(\mathbf{x})-\mu)}{\sigma}\right)^{-1/\xi} P(y')}. \tag{7.114}$$

This formulation ensures that as $L(\mathbf{x})$ moves further into the tail, the posterior probability of all known classes decreases, leading to a natural rejection mechanism for OOD samples. An alternative approach to likelihood estimation involves using a logistic function to approximate the EVT threshold behavior, given by

$$P(\mathbf{x}|y) \approx \frac{1}{1 + e^{-\alpha(L(\mathbf{x})-\tau)}} \tag{7.115}$$

where α is a scale parameter controlling the steepness of the probability drop-off. The EVT-based model ensures that open set detection is grounded in rigorous statistical principles, as opposed to ad-hoc thresholding methods. The theoretical justification of EVT-based likelihood estimation follows from the asymptotic stability of the GPD under maximum domain of attraction conditions, ensuring that the fitted distribution remains valid for new samples. Consequently, the EVT-based formulation provides a robust and theoretically sound approach to estimating likelihoods under generative models, enabling principled open set recognition.

7.7 Bayesian Neural Networks (BNNs)

In generative modeling, the open set learning problem is fundamentally framed in terms of estimating the likelihood $P(x|y)$ under Bayesian Neural Networks (BNNs) because the Bayesian framework allows for a principled quantification of uncertainty. Open set learning requires the model to be aware of its epistemic uncertainty—uncertainty arising from a lack of knowledge about the data distribution. This is crucial for distinguishing between in-distribution (ID) data and out-of-distribution (OOD) data, a problem that traditional deterministic neural networks struggle with. The Bayesian formulation mitigates this issue by considering an entire distribution over model parameters θ, rather than a single point estimate. The posterior distribution over the parameters given the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ is given by Bayes' theorem as

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}, \tag{7.116}$$

where $P(\mathcal{D}|\theta)$ is the likelihood function, $P(\theta)$ is the prior over the network parameters, and $P(\mathcal{D})$ is the marginal likelihood (also known as the model evidence), which ensures proper normalization. Since the marginal likelihood is intractable in deep neural networks due to the high-dimensional integral over all possible parameter configurations, approximate inference techniques such as variational inference, Monte Carlo methods (e.g., Hamiltonian Monte Carlo or Stochastic Gradient Langevin Dynamics), or Laplace approximations are employed to obtain an approximation to the posterior distribution. Given this posterior, the Bayesian predictive distribution for a new input x^* conditioned on a class label y^* is obtained by marginalizing over θ:

$$P(x^*|y^*) = \int P(x^*|y^*, \theta)P(\theta|\mathcal{D})\, d\theta. \tag{7.117}$$

This integral encodes both aleatoric uncertainty (due to inherent noise in the data) and epistemic uncertainty (due to lack of knowledge). Since this integral is computationally intractable for high-dimensional parameter spaces, it is typically approximated via Monte Carlo integration using an ensemble of sampled parameters $\theta_m \sim P(\theta|\mathcal{D})$:

$$P(x^*|y^*) \approx \frac{1}{M}\sum_{m=1}^{M} P(x^*|y^*, \theta_m). \tag{7.118}$$

A fundamental property of Bayesian models in the context of open set learning is that the variance of the predictive distribution provides a direct measure of epistemic uncertainty. The predictive mean and variance are given by

$$\mathbb{E}[x^*|y^*, \mathcal{D}] = \int x^* P(x^*|y^*, \theta)P(\theta|\mathcal{D})\, d\theta, \tag{7.119}$$

$$\mathrm{Var}[x^*|y^*, \mathcal{D}] = \int x^{*2} P(x^*|y^*, \theta)P(\theta|\mathcal{D})\, d\theta - (\mathbb{E}[x^*|y^*, \mathcal{D}])^2. \tag{7.120}$$

For inputs x^* that are outside the training distribution, the variance of the predictive distribution is expected to be significantly higher because the posterior over θ is conditioned only on the observed training data and lacks information about unseen samples. This property allows Bayesian models to naturally detect OOD data by thresholding on predictive uncertainty. To model the conditional likelihood $P(x|y)$, generative Bayesian models introduce latent variables z to capture underlying structure in the data:

$$P(x|y, \theta) = \int P(x|z, \theta)P(z|y, \theta)\, dz. \tag{7.121}$$

Since direct inference over the latent variable posterior $P(z|x, y, \theta)$ is intractable, variational inference approximates it with a variational distribution $q(z|x, y, \phi)$, leading to the Evidence Lower Bound (ELBO):

$$\log P(x|y) \geq \mathbb{E}_{q(z|x,y,\phi)}\left[\log P(x|z, \theta)\right] - D_{\mathrm{KL}}(q(z|x, y, \phi)||P(z|y, \theta)). \tag{7.122}$$

Here, $D_{\mathrm{KL}}(q||p)$ denotes the Kullback-Leibler divergence, which enforces regularization by minimizing the difference between the approximate posterior and the true prior. In practice, deep generative models such as Variational Autoencoders (VAEs), Normalizing Flows, and Deep Energy-Based Models leverage Bayesian formulations to learn structured representations of $P(x|y)$ in high-dimensional spaces. From a Bayesian decision-theoretic perspective, the total uncertainty in a prediction is measured using the predictive entropy:

$$H(x^*|y^*) = -\int P(x^*|y^*, \theta) \log P(x^*|y^*, \theta)\, d\theta. \tag{7.123}$$

Decomposing this entropy into aleatoric and epistemic components via mutual information, the epistemic uncertainty is captured by the expected Kullback-Leibler divergence between individual parameter-conditioned predictions and the full posterior predictive distribution:

$$\mathrm{MI}[x^*, \theta|y^*] = \mathbb{E}_{P(\theta|\mathcal{D})} \left[D_{\mathrm{KL}}(P(x^*|y^*, \theta) || P(x^*|y^*)) \right]. \tag{7.124}$$

Higher epistemic uncertainty indicates that the model is less confident in its prediction due to insufficient training data, which is a key indicator of an OOD sample. By setting an uncertainty threshold, a Bayesian classifier can reject predictions on OOD data, effectively implementing an open set classifier.

Bayesian neural networks thus provide a mathematically rigorous solution to the open set learning problem by naturally incorporating uncertainty through posterior inference over model parameters, leveraging predictive distributions to measure epistemic uncertainty, and applying principled probabilistic inference techniques such as variational approximations and Monte Carlo integration. The estimation of $P(x|y)$ in this framework allows the model to dynamically adjust its confidence in predictions and provide robust detection of OOD samples by analyzing likelihood distributions.

7.8 Support Vector Models

In the context of generative modeling, the open set learning problem is rigorously formalized as the estimation of the conditional likelihood $P(x|y)$ under Support Vector Machines (SVMs) by employing advanced probabilistic frameworks, optimization theory, and statistical learning principles. The objective is to model the distribution of data x conditioned on class labels y, where y may belong to a known set of classes $\mathcal{Y} = \{1, 2, \ldots, K\}$ or an unknown class $y \notin \mathcal{Y}$. This necessitates a mathematically precise formulation that integrates discriminative learning with probabilistic reasoning to handle both known and unknown classes. The likelihood $P(x|y)$ is derived from Bayes' theorem, which decomposes the conditional probability as:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}, \tag{7.125}$$

where $P(y|x)$ is the posterior probability of class y given x, $P(x)$ is the marginal distribution of x, and $P(y)$ is the prior probability of class y. Under the SVM framework, the posterior $P(y|x)$ is modeled using a discriminant function $f_y(x)$, which is optimized to separate classes while maximizing the margin. The optimization problem for learning $f_y(x)$ is formulated as:

$$\min_{f_y, \xi} \frac{1}{2} \|f_y\|_{\mathcal{H}}^2 + C \sum_{i=1}^{N} \xi_i, \tag{7.126}$$

subject to the constraints:

$$y_i f_y(x_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, N, \tag{7.127}$$

where $\|f_y\|_{\mathcal{H}}$ denotes the norm of f_y in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, C is a regularization parameter controlling the trade-off between margin maximization and classification error, and ξ_i are slack variables accounting for misclassifications. The discriminant function $f_y(x)$ is expressed as:

$$f_y(x) = \mathbf{w}_y^T \phi(x) + b_y, \tag{7.128}$$

where $\mathbf{w}_y$ is the weight vector in the feature space, $\phi(x)$ is a feature mapping function, and b_y is the bias term. To extend this framework to open set learning, the likelihood $P(x|y)$ is augmented with a rejection mechanism to handle unknown classes. This is achieved by introducing a class-specific threshold τ_y, such that:

$$P(x|y) = \begin{cases} \frac{P(y|x)P(x)}{P(y)} & \text{if } f_y(x) \geq \tau_y, \\ 0 & \text{otherwise.} \end{cases} \tag{7.129}$$

The threshold τ_y is determined by minimizing the empirical risk on a validation set $\mathcal{D}_{\text{val}} = \{(x_i, y_i)\}_{i=1}^M$, formalized as:

$$\tau_y = \arg\min_{\tau} \sum_{i=1}^{M} \mathbb{I}(f_y(x_i) < \tau) \cdot L(y_i, \text{unknown}), \tag{7.130}$$

where $\mathbb{I}(\cdot)$ is the indicator function, and $L(y_i, \text{unknown})$ is the loss incurred when an instance from an unknown class is misclassified as belonging to class y. This ensures that the model rejects instances from unknown classes with high confidence. The likelihood $P(x|y)$ is further refined using kernel methods, which enable SVMs to operate in a high-dimensional feature space. The kernel function $k(x, x')$ is defined as:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}, \tag{7.131}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in the RKHS $\mathcal{H}$. The discriminant function $f_y(x)$ in the kernelized form is given by:

$$f_y(x) = \sum_{i=1}^{N} \alpha_i y_i k(x_i, x) + b_y, \tag{7.132}$$

where α_i are the Lagrange multipliers obtained from solving the dual optimization problem:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.133}$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N} \alpha_i y_i = 0. \tag{7.134}$$

The kernelized discriminant function allows the model to capture complex decision boundaries in the input space. The likelihood $P(x|y)$ is then estimated by combining the kernelized discriminant function with the rejection mechanism:

$$P(x|y) = \begin{cases} \frac{\exp(f_y(x))}{\sum_{y' \in \mathcal{Y}} \exp(f_{y'}(x))} & \text{if } f_y(x) \geq \tau_y, \\ 0 & \text{otherwise.} \end{cases} \tag{7.135}$$

This formulation ensures that the model assigns high likelihood to instances from known classes while rejecting instances from unknown classes. The overall objective is to maximize the log-likelihood of the observed data under the model:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log P(x_i | y_i; \theta), \tag{7.136}$$

where θ represents the parameters of the model, including $\mathbf{w}_y$, b_y, and τ_y. To further enhance the rigor, the optimization problem is regularized using a penalty term $\Omega(\theta)$ to prevent overfitting:

$$\min_{\theta} -\mathcal{L}(\theta) + \lambda \Omega(\theta), \tag{7.137}$$

where λ is a regularization parameter, and $\Omega(\theta)$ is typically chosen as the L_2-norm of the parameters:

$$\Omega(\theta) = \|\mathbf{w}_y\|_2^2 + \|b_y\|_2^2. \tag{7.138}$$

This ensures that the model generalizes well to unseen data while maintaining discriminative power.

In conclusion, the open set learning problem under SVMs is rigorously framed as the estimation of $P(x|y)$ through a combination of probabilistic modeling, kernel methods, and optimization. The discriminant function $f_y(x)$ is optimized to separate known classes while incorporating a rejection mechanism for unknown classes, ensuring robust performance in open set scenarios. The likelihood $P(x|y)$ is derived using Bayes' theorem and refined through kernelized discriminant functions, resulting in a mathematically precise and computationally tractable framework for open set learning.

7.9 Support Vector Data Description

In the context of generative modeling, the open set learning problem is rigorously formalized under the Support Vector Data Description (SVDD) framework by constructing a probabilistic model for the likelihood $P(x|y)$ through the optimization of a hypersphere that encapsulates the data distribution of class y. The hypersphere is defined by its center $\mathbf{a} \in \mathbb{R}^d$ and radius $R \in \mathbb{R}_+$, which are determined by solving a constrained optimization problem that minimizes the volume of the sphere while ensuring that the majority of the training data $\mathbf{x}_i \in \mathcal{X}$ for class y lie within or close to the sphere. The primal optimization problem is formulated as:

$$\min_{R,\mathbf{a},\xi_i} R^2 + C\sum_{i=1}^{N} \xi_i, \tag{7.139}$$

subject to the constraints:

$$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, 2, \ldots, N, \tag{7.140}$$

where $\xi_i \in \mathbb{R}_+$ are slack variables that permit deviations from the strict boundary, $C \in \mathbb{R}_+$ is a regularization parameter that balances the trade-off between the sphere's volume and the penalty for outliers, and N is the cardinality of the training set. The likelihood $P(x|y)$ is modeled as a function of the squared Euclidean distance $\|\mathbf{x} - \mathbf{a}\|^2$, which quantifies the deviation of a test point $\mathbf{x}$ from the center $\mathbf{a}$. Specifically, the likelihood is expressed as:

$$P(x|y) = \frac{1}{Z} \exp\left(-\frac{\|\mathbf{x} - \mathbf{a}\|^2}{\sigma^2}\right), \tag{7.141}$$

where Z is a normalization constant ensuring that $\int P(x|y)\, d\mathbf{x} = 1$, and $\sigma \in \mathbb{R}_+$ is a scaling parameter that governs the rate of decay of the likelihood as the distance from the center increases. The distance $\|\mathbf{x} - \mathbf{a}\|^2$ is computed in a feature space induced by a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which allows for non-linear decision boundaries. The kernelized distance is given by:

$$\|\mathbf{x} - \mathbf{a}\|^2 = K(\mathbf{x}, \mathbf{x}) - 2\sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{7.142}$$

where $\alpha_i \in \mathbb{R}_+$ are the Lagrange multipliers obtained from the dual formulation of the SVDD optimization problem. The dual problem is derived by introducing Lagrange multipliers α_i and γ_i for the inequality constraints, resulting in the following Lagrangian:

$$\mathcal{L}(R, \mathbf{a}, \xi_i, \alpha_i, \gamma_i) = R^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left(R^2 + \xi_i - \|\mathbf{x}_i - \mathbf{a}\|^2\right) - \sum_{i=1}^{N} \gamma_i \xi_i. \tag{7.143}$$

By setting the derivatives of the Lagrangian with respect to R, $\mathbf{a}$, and ξ_i to zero, the dual optimization problem is obtained as:

$$\max_{\alpha_i} \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{7.144}$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N} \alpha_i = 1. \tag{7.145}$$

The solution to the dual problem yields the Lagrange multipliers α_i, which are used to compute the center $\mathbf{a}$ as:

$$\mathbf{a} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i. \tag{7.146}$$

The radius R is determined by selecting any support vector $\mathbf{x}_k$ for which $0 < \alpha_k < C$, and computing:

$$R^2 = \|\mathbf{x}_k - \mathbf{a}\|^2 = K(\mathbf{x}_k, \mathbf{x}_k) - 2\sum_{i=1}^{N} \alpha_i K(\mathbf{x}_k, \mathbf{x}_i) + \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{7.147}$$

The likelihood $P(x|y)$ is then used to classify a test point $\mathbf{x}$ as belonging to the open set if the likelihood falls below a threshold $\tau \in \mathbb{R}_+$, which is determined based on the desired trade-off between false positives and false negatives. The decision rule is formalized as:

$$\text{Classify } \mathbf{x} \text{ as open set if } P(x|y) < \tau. \tag{7.148}$$

The threshold τ can be estimated using cross-validation or other statistical techniques to optimize the model's performance. This formulation provides a rigorous and mathematically sound framework for open set learning, combining principles from optimization, kernel methods, and probabilistic modeling to estimate the likelihood $P(x|y)$ and distinguish between known and unknown classes in a principled manner. The SVDD-based approach ensures that the model generalizes well to unseen data while maintaining computational efficiency and robustness to outliers.

8 Zero-Shot Learning

Zero-Shot learning (ZSL) is a highly sophisticated paradigm in machine learning wherein a model is expected to classify instances from classes it has never encountered during training. Mathematically, we define a set of seen classes S and unseen classes U, such that the training set consists of labeled instances (x, y) where $y \in S$, and the test set consists of instances x' where $y' \in U$, with $S \cap U = \emptyset$. The challenge of ZSL lies in leveraging semantic knowledge to generalize beyond the training distribution, necessitating a mapping function $f : X \to Y$ that extends to unseen categories.

The problem can be formulated using an attribute-based representation, where each class y is associated with a high-dimensional semantic embedding $a(y) \in \mathbb{R}^d$, which encodes meaningful relationships between classes. The objective is to learn a function $g : X \to \mathbb{R}^d$ that maps input samples to the same semantic space. Given a new sample x', the predicted class is determined by minimizing a distance function d:

$$\hat{y} = \arg\min_{y \in U} d(g(x'), a(y)) \tag{8.1}$$

A common choice for d is the squared Euclidean distance:

$$d(g(x'), a(y)) = \|g(x') - a(y)\|^2 \tag{8.2}$$

or the cosine similarity:

$$d(g(x'), a(y)) = 1 - \frac{\langle g(x'), a(y) \rangle}{\|g(x')\| \|a(y)\|} \tag{8.3}$$

The function g is often parameterized by a neural network with parameters θ, trained via minimizing a contrastive loss:

$$\mathcal{L}(\theta) = \sum_{(x,y) \in S} \|g_\theta(x) - a(y)\|^2 \tag{8.4}$$

or a cross-entropy loss in a transformed space:

$$\mathcal{L}(\theta) = - \sum_{(x,y) \in S} \log \frac{e^{\phi(g_\theta(x), a(y))}}{\sum_{y' \in S} e^{\phi(g_\theta(x), a(y'))}} \tag{8.5}$$

where ϕ is a similarity measure, often a bilinear transformation:

$$\phi(g(x), a(y)) = g(x)^T W a(y) \tag{8.6}$$

for some learned weight matrix W. Generalization to unseen classes depends on the structure of the semantic space, typically constructed using linguistic embeddings (e.g., Word2Vec, GloVe) or manually defined attribute vectors. The effectiveness of ZSL hinges on the assumption that semantic relationships transfer across classes, which can be formalized through a probabilistic model:

$$P(y|x) \propto P(x|a(y)) P(a(y)) \tag{8.7}$$

where $P(x|a(y))$ is modeled via a generative process:

$$P(x|a(y)) = \int P(x|z)P(z|a(y))dz \tag{8.8}$$

with z representing latent features shared across seen and unseen classes. Variational inference is commonly employed to approximate this integral, leading to an evidence lower bound (ELBO):

$$\log P(x|a(y)) \geq \mathbb{E}_{q(z|x)}\left[\log P(x|z)\right] - D_{\mathrm{KL}}(q(z|x)\|P(z|a(y))) \tag{8.9}$$

where $q(z|x)$ is an inference network. Zero-shot learning can also be formulated through a linear mapping W such that:

$$g(x) = Wf(x) \tag{8.10}$$

where $f(x)$ is a feature extractor (e.g., a deep CNN), and W aligns it with the semantic space. This leads to a ridge regression formulation:

$$W^* = \arg\min_W \sum_{(x,y)\in S} \|Wf(x) - a(y)\|^2 + \lambda\|W\|^2 \tag{8.11}$$

solvable via the closed-form solution:

$$W^* = AF^T(FF^T + \lambda I)^{-1} \tag{8.12}$$

where A is the matrix of attribute vectors and F is the matrix of extracted features. The zero-shot classification task then reduces to:

$$\hat{y} = \arg\max_{y\in U} g(x')^T a(y) \tag{8.13}$$

which assigns the test instance to the nearest class in the semantic space. Generalized zero-shot learning (GZSL) extends this by allowing both seen and unseen classes at test time, requiring a calibrated compatibility function:

$$\hat{y} = \arg\max_{y\in S\cup U} \left[g(x')^T a(y) - \gamma\mathbb{I}(y \in S)\right] \tag{8.14}$$

where γ controls the bias toward seen classes. The choice of semantic space and feature extractor significantly affects performance, as different embedding choices induce different generalization behaviors. Alternative formulations include energy-based models:

$$E(x, y) = g(x)^T Wa(y) + b_y \tag{8.15}$$

trained via hinge loss:

$$\mathcal{L} = \sum_{(x,y)\in S} \sum_{y'\neq y} \max(0, 1 - E(x, y) + E(x, y')) \tag{8.16}$$

or meta-learning approaches that define a task-specific adaptation function:

$$\theta' = \theta - \alpha\nabla_\theta \mathcal{L}_{\mathrm{meta}}(X_{\mathrm{train}}, Y_{\mathrm{train}}) \tag{8.17}$$

to optimize generalization. These diverse formulations illustrate the intricate mathematical foundation of zero-shot learning, where the interplay between feature extraction, semantic embedding, and transfer mechanisms defines the success of the model in recognizing previously unseen categories.

8.1 Energy-based models for Zero-Shot Learning

Energy-based models (EBMs) for zero-shot learning (ZSL) provide a mathematically rigorous framework for learning representations that generalize beyond seen categories by modeling the compatibility between data samples and their semantic attributes. The core principle of EBMs in ZSL is the minimization of an energy function $E(x, y)$, which defines a measure of compatibility between an input sample x and a class label y. This function is designed such that correct pairings of (x, y) have lower energy values compared to incorrect pairings. The probability of assigning x to class y is given by the Boltzmann distribution:

$$P(y|x) = \frac{\exp(-E(x, y))}{\sum_{y'} \exp(-E(x, y'))} \tag{8.18}$$

where the denominator represents the partition function, ensuring normalization across all possible class labels. This formulation enforces a probabilistic interpretation of the energy function, ensuring that samples are more likely assigned to classes with lower energy. In zero-shot learning, the fundamental challenge is to extend knowledge to unseen classes using auxiliary information, often in the form of semantic embeddings $a(y)$, which encode prior knowledge about each class y. The energy function is typically parameterized as a compatibility function between the input x and the semantic representation $a(y)$, formulated as

$$E(x, y) = -f(x)^T W a(y) \tag{8.19}$$

where $f(x)$ is a feature extractor mapping x into a latent space, and W is a learnable weight matrix. The objective function is constructed to minimize the energy for correct class-attribute associations while maximizing it for incorrect ones. A contrastive loss is often used to enforce this property:

$$\mathcal{L} = \sum_{(x,y)} \left[E(x, y) + \log \sum_{y' \neq y} \exp(-E(x, y')) \right] \tag{8.20}$$

which penalizes incorrect associations by ensuring the energy of the correct class y is lower than all incorrect alternatives y'. To enhance generalization, additional regularization terms are often introduced. A common approach is the margin-based loss:

$$\mathcal{L}_{\text{margin}} = \sum_{(x,y)} \sum_{y' \neq y} \max(0, \Delta + E(x, y) - E(x, y')) \tag{8.21}$$

where Δ is a margin enforcing separation between correct and incorrect classes. This formulation ensures robustness against overfitting to seen classes and enables transfer to unseen classes. To improve stability, energy-based models often employ gradient-based optimization strategies, leveraging backpropagation to refine W, $f(x)$, and other parameters. The gradients are computed as

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{(x,y)} \left[\frac{\partial E(x, y)}{\partial W} - \sum_{y'} P(y'|x) \frac{\partial E(x, y')}{\partial W} \right] \tag{8.22}$$

$$\frac{\partial \mathcal{L}}{\partial f(x)} = \sum_{(x,y)} \left[\frac{\partial E(x, y)}{\partial f(x)} - \sum_{y'} P(y'|x) \frac{\partial E(x, y')}{\partial f(x)} \right] \tag{8.23}$$

where the expectation over incorrect classes ensures optimization is guided towards reducing energy for true class-attribute associations. A crucial aspect of EBMs in ZSL is the use of generative constraints, where the model is regularized using a reconstruction loss to ensure feature consistency across seen and unseen categories. A common approach involves a variational autoencoder (VAE)-based loss:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)} \left[-E(x, y) + \lambda D_{\text{KL}}(q(z|x) \parallel p(z)) \right] \tag{8.24}$$

where $q(z|x)$ is an approximate posterior, and $p(z)$ is a prior distribution. This enforces meaningful latent representations for generalization to unseen classes. To further refine the learned embeddings, contrastive self-supervised techniques can be integrated. The contrastive learning loss is defined as

$$\mathcal{L}_{\text{contrastive}} = \sum_{x_i, x_j \sim \mathcal{P}(x)} -\log \frac{\exp(E(x_i, x_j)/\tau)}{\sum_{x_k} \exp(E(x_i, x_k)/\tau)} \tag{8.25}$$

where τ is a temperature parameter and $\mathcal{P}(x)$ is a positive sample pair distribution. This regularization aligns semantically similar instances, ensuring the energy function learns robust and transferable representations. The inference stage in zero-shot learning using EBMs involves solving an optimization problem to assign a query sample x to the class y that minimizes the energy function:

$$\hat{y} = \arg\min_{y} E(x, y) \tag{8.26}$$

where $E(x, y)$ is evaluated for both seen and unseen classes. Since the energy function is trained using both contrastive and probabilistic constraints, it generalizes effectively to novel categories. In summary, EBMs for ZSL leverage compatibility functions, contrastive constraints, and probabilistic formulations to ensure robust zero-shot generalization. The interplay between energy minimization, semantic embeddings, and self-supervised learning techniques provides a mathematically rigorous approach for knowledge transfer across seen and unseen categories.

8.1.1 Various Loss function used in Energy-based models for Zero-Shot Learning

Contrastive loss in Energy-based Models (EBMs) for Zero-Shot Learning (ZSL) is designed to ensure that embeddings of similar instances are brought closer together, while embeddings of dissimilar instances are pushed apart. Given a set of input-label pairs (x, y), the energy function $E(x, y)$ is minimized for correct pairs and maximized for incorrect ones. The contrastive loss function can be formulated as

$$\mathcal{L}_{\text{contrastive}} = \sum_{(x,y)} \left[\mathbb{I}(y = y^+) E(x, y^+) + \mathbb{I}(y \neq y^+) \max(0, m - E(x, y^-)) \right] \tag{8.27}$$

where y^+ represents the true label, y^- represents a negative label, $\mathbb{I}(\cdot)$ is an indicator function, and m is a margin parameter ensuring sufficient separation between positive and negative pairs. The energy function $E(x, y)$ is typically parameterized using neural networks and optimized via stochastic gradient descent. The gradient updates follow

$$\nabla_\theta \mathcal{L}_{\text{contrastive}} = \sum_{(x,y)} \left[\mathbb{I}(y = y^+) \nabla_\theta E(x, y^+) - \mathbb{I}(y \neq y^+) \nabla_\theta \max(0, m - E(x, y^-)) \right]. \tag{8.28}$$

Margin-based loss extends the contrastive loss by enforcing a stricter separation criterion between positive and negative pairs. The loss function is

$$\mathcal{L}_{\text{margin}} = \sum_{(x,y)} \left[E(x, y^+) - E(x, y^-) + m \right]_+ \tag{8.29}$$

where $[z]_+ = \max(0, z)$ ensures non-negative loss contributions. This formulation encourages $E(x, y^+)$ to be strictly lower than $E(x, y^-)$ by at least margin m. The gradient computation follows

$$\nabla_\theta \mathcal{L}_{\text{margin}} = \sum_{(x,y)} \mathbb{I}(E(x, y^-) - E(x, y^+) < m) \left[\nabla_\theta E(x, y^+) - \nabla_\theta E(x, y^-) \right]. \tag{8.30}$$

Variational Autoencoder (VAE)-based loss introduces probabilistic modeling into EBMs for ZSL by defining a latent variable z that captures the underlying structure of the data. The energy function is parameterized as

$$E(x, y, z) = -\log p(x|z) - \log p(z|y) - \log p(y). \tag{8.31}$$

The VAE-based loss is the negative Evidence Lower Bound (ELBO), given by

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)}\left[E(x, y, z)\right] - D_{\text{KL}}(q(z|x)\|p(z|y)), \tag{8.32}$$

where $q(z|x)$ is the approximate posterior, $p(z|y)$ is the prior, and D_{KL} denotes the Kullback-Leibler divergence. The gradient updates for training involve

$$\nabla_\theta \mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)}\left[\nabla_\theta E(x, y, z)\right] - \nabla_\theta D_{\text{KL}}(q(z|x)\|p(z|y)). \tag{8.33}$$

Contrastive learning loss in EBMs for ZSL builds upon the contrastive framework by explicitly constructing positive and negative pairs in an embedding space where similarity is measured. A common formulation is the InfoNCE loss

$$\mathcal{L}_{\text{contrastive learning}} = -\mathbb{E}_{(x,y)} \log \frac{\exp(-E(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(-E(x, y'))}. \tag{8.34}$$

This formulation ensures that the energy of the correct label $E(x, y)$ is minimized while normalizing across all possible labels. The corresponding gradient updates are

$$\nabla_\theta \mathcal{L}_{\text{contrastive learning}} = \mathbb{E}_{(x,y)}\left[\nabla_\theta E(x, y) - \sum_{y' \in \mathcal{Y}} p(y'|x) \nabla_\theta E(x, y')\right], \tag{8.35}$$

where $p(y'|x)$ is the probability of a label under the energy model. Each of these loss functions plays a crucial role in shaping the energy landscape for Zero-Shot Learning, ensuring effective generalization to unseen classes.

8.1.2 Generative constraints in Energy-based models for Zero-Shot Learning

Generative constraints in energy-based models (EBMs) for zero-shot learning (ZSL) fundamentally stem from the principle of defining an energy landscape that enforces semantic compatibility between seen and unseen classes while maintaining representational fidelity to underlying data distributions. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^d$ represents input features and $y_i \in \mathcal{Y}_S$ belongs to the set of seen classes, the primary objective is to infer meaningful representations for an unseen class set $\mathcal{Y}_U$, such that $\mathcal{Y}_S \cap \mathcal{Y}_U = \emptyset$. The generative constraints in EBMs introduce an energy function $E(x, y; \theta)$, parameterized by θ, that assigns low energy to compatible (x, y) pairs and high energy to incompatible ones, ensuring that the generative structure generalizes across class distributions.

$$E(x, y; \theta) = -f_\theta(x)^\top g_\theta(y), \tag{8.36}$$

where $f_\theta(x)$ is the feature extractor mapping inputs to an embedding space, and $g_\theta(y)$ is the semantic mapping function capturing class-level knowledge. Zero-shot generalization is imposed via a generative constraint that enforces energy-based regularization between seen and unseen class distributions, expressed as

$$\mathbb{E}_{(x,y)\sim p_S(x,y)}[E(x, y; \theta)] \approx \mathbb{E}_{(x,y)\sim p_U(x,y)}[E(x, y; \theta)], \tag{8.37}$$

where $p_S(x, y)$ and $p_U(x, y)$ are the joint distributions over seen and unseen data, respectively. This constraint ensures that the energy landscape maintains generalizability by aligning unseen class embeddings with seen class statistics. The generative constraints can be explicitly incorporated via an adversarial loss term that minimizes the Kullback-Leibler (KL) divergence between seen and unseen class distributions in the latent space:

$$\mathcal{L}_{\text{gen}} = D_{\text{KL}}(p_U(z) \| p_S(z)), \tag{8.38}$$

where $z = f_\theta(x)$ represents the latent feature encoding. Since direct estimation of $p_U(z)$ is intractable, it is typically approximated using a generative model $q_\phi(z|y)$, leading to the variational formulation:

$$\mathcal{L}_{\text{gen}} = \mathbb{E}_{y \sim p_U(y)} \left[\mathbb{E}_{z \sim q_\phi(z|y)} \log \frac{q_\phi(z|y)}{p_S(z)} \right]. \tag{8.39}$$

This enforces a shared latent space between seen and unseen distributions, facilitating knowledge transfer in a generative manner. A key generative constraint in EBMs is the entropy-based regularization, which ensures that the energy landscape does not collapse into degenerate solutions. The entropy of the conditional distribution $p(y|x;\theta) \propto e^{-E(x,y;\theta)}$ is maximized via

$$\mathcal{L}_{\text{entropy}} = -\mathbb{E}_{p(x)} \sum_y p(y|x;\theta) \log p(y|x;\theta), \tag{8.40}$$

which prevents the model from overfitting to seen classes by enforcing a smooth energy manifold. This is further complemented by a contrastive loss enforcing inter-class separability:

$$\mathcal{L}_{\text{contrast}} = \mathbb{E}_{(x,y) \sim p_S(x,y)} \sum_{y' \neq y} e^{E(x,y';\theta) - E(x,y;\theta)}. \tag{8.41}$$

Together, these constraints regularize the energy function such that the generative structure captures semantic transferability between seen and unseen categories. Another fundamental generative constraint involves aligning class prototypes with their corresponding semantic representations through a structural loss

$$\mathcal{L}_{\text{proto}} = \sum_{y \in \mathcal{Y}_S \cup \mathcal{Y}_U} \| g_\theta(y) - \mathbb{E}_{p(x|y)}[f_\theta(x)] \|^2. \tag{8.42}$$

This ensures that the learned embeddings remain consistent with high-level class descriptions, enabling robust generalization to unseen categories. A further refinement of the generative constraint is the incorporation of a marginal energy regularization term, enforcing stability across variations in input perturbations:

$$\mathcal{L}_{\text{marginal}} = \mathbb{E}_{(x,y) \sim p_S(x,y)} \left[\max_{x' \sim \mathcal{B}(x)} \big(E(x', y; \theta) - E(x, y; \theta) \big) \right], \tag{8.43}$$

where $\mathcal{B}(x)$ denotes a local perturbation neighborhood of x. This encourages energy invariance within class-consistent transformations, further reinforcing generalization properties. Finally, the overall training objective for EBMs in zero-shot learning integrates the aforementioned generative constraints into a single optimization problem:

$$\min_\theta \mathcal{L}_{\text{total}} = \mathcal{L}_{\text{contrast}} + \lambda_1 \mathcal{L}_{\text{gen}} + \lambda_2 \mathcal{L}_{\text{entropy}} + \lambda_3 \mathcal{L}_{\text{proto}} + \lambda_4 \mathcal{L}_{\text{marginal}}, \tag{8.44}$$

where $\lambda_1, \lambda_2, \lambda_3$, and λ_4 are hyperparameters balancing the contributions of different generative constraints. This structured formulation ensures that the learned energy function captures meaningful relationships across both seen and unseen classes, enabling zero-shot generalization through robust energy-based generative modeling.

8.1.3 Use of Inference in Energy-based models for Zero-Shot Learning

Energy-based models (EBMs) provide a principled framework for zero-shot learning (ZSL) by defining a scalar energy function $E(x, y)$ that assigns compatibility scores to input data samples x and output labels y. The core idea of inference in energy-based models for zero-shot learning is to leverage a structured energy landscape that captures semantic relationships between seen and unseen classes. This enables generalization to novel classes without requiring explicit training examples for them. The inference process is governed by the minimization of energy functions that encode both data likelihood and constraints imposed by semantic knowledge transfer mechanisms.

Mathematically, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ consisting of training examples with inputs $x_i \in \mathcal{X}$ and their corresponding class labels $y_i \in \mathcal{Y}_{\text{seen}}$, the goal of zero-shot inference is to predict labels $y^* \in \mathcal{Y}_{\text{unseen}}$ for new inputs. The energy function is typically defined as

$$E(x, y) = -f_\theta(x, y) \tag{8.45}$$

where $f_\theta(x, y)$ is a learnable compatibility function parameterized by θ. Inference in EBMs for ZSL is performed by solving the minimization problem

$$y^* = \arg \min_{y \in \mathcal{Y}_{\text{unseen}}} E(x, y). \tag{8.46}$$

The energy function is often decomposed into different terms capturing multiple constraints, such as semantic consistency, visual-semantic alignment, and regularization. A common formulation incorporates a bilinear compatibility function between input feature representations $\phi(x)$ and class embeddings $\psi(y)$:

$$E(x, y) = -\phi(x)^T W \psi(y), \tag{8.47}$$

where W is a learnable weight matrix encoding the mapping between the visual and semantic spaces. The inference process then reduces to

$$y^* = \arg \max_{y \in \mathcal{Y}_{\text{unseen}}} \phi(x)^T W \psi(y). \tag{8.48}$$

Energy minimization for inference can also be formulated as a probabilistic model using the Boltzmann distribution:

$$P(y|x) = \frac{\exp(-E(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(-E(x, y'))}. \tag{8.49}$$

Inference corresponds to maximum a posteriori (MAP) estimation:

$$y^* = \arg \max_{y \in \mathcal{Y}_{\text{unseen}}} P(y|x). \tag{8.50}$$

One approach to improve inference performance in ZSL is to introduce additional constraints through a margin-based ranking loss:

$$\sum_{(x,y) \in \mathcal{D}} \sum_{y' \neq y} \max(0, E(x, y) - E(x, y') + \Delta), \tag{8.51}$$

where Δ is a margin parameter ensuring that the correct label has a lower energy than incorrect ones. In scenarios where inference involves generative modeling, energy-based models can be coupled with contrastive divergence training. The gradient of the energy function with respect to the input features provides a principled way to update representations:

$$\frac{\partial E(x, y)}{\partial x} = -\frac{\partial f_\theta(x, y)}{\partial x}. \tag{8.52}$$

By leveraging learned semantic embeddings, energy minimization enables inference even in the absence of explicit training data for unseen categories. The effectiveness of energy-based models for zero-shot learning thus fundamentally relies on the quality of the learned energy landscape, ensuring smooth generalization from seen to unseen classes through structured energy minimization.

8.2 Meta-learning approaches for Zero-Shot Learning

Meta-learning approaches for zero-shot learning (ZSL) are fundamentally based on the principle of learning to learn, where a model is trained on a set of tasks such that it can generalize to novel tasks it has never encountered before. The primary challenge in zero-shot learning lies in the fact that during inference, the model must classify data samples from categories that were not seen during training. Meta-learning addresses this challenge by acquiring transferable knowledge that allows for rapid adaptation to new categories using minimal additional information. Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ the label space. Given a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{train}}}$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}_{\text{train}}$, the objective in zero-shot learning is to classify a test sample $\mathbf{x}^*$ into one of the unseen categories $\mathcal{Y}_{\text{test}}$, such that $\mathcal{Y}_{\text{train}} \cap \mathcal{Y}_{\text{test}} = \emptyset$.

Meta-learning operates by optimizing a function f_θ parameterized by θ such that it minimizes an expected loss over a distribution of tasks. Each task $\mathcal{T}_i$ is drawn from a distribution $p(\mathcal{T})$, where a task consists of a small support set $S_i = \{(\mathbf{x}_j, y_j)\}_{j=1}^{k}$ and a query set $Q_i = \{(\mathbf{x}_j, y_j)\}_{j=1}^{m}$. The meta-objective is to minimize the expected loss over all tasks:

$$\mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}(f_\theta, S, Q) \right]. \tag{8.53}$$

A common meta-learning paradigm is the Model-Agnostic Meta-Learning (MAML) approach, which seeks to find an initial set of parameters θ such that fine-tuning on a new task requires minimal adaptation. Given a task $\mathcal{T}_i$ with support set S_i, MAML updates θ using one or more gradient descent steps:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta), \tag{8.54}$$

where α is the learning rate. The meta-update then optimizes θ by minimizing the loss over query samples after adaptation:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}), \tag{8.55}$$

where β is the meta-learning rate. Another approach is the meta-embedding strategy, where feature representations are learned to generalize across tasks. Let $\phi(\mathbf{x}; \theta)$ denote an embedding function mapping inputs to a latent space $\mathbb{R}^d$. The objective is to learn an embedding that maintains a consistent similarity structure across both seen and unseen categories. The compatibility function $F(\mathbf{x}, y; \theta)$ is often modeled as a bilinear compatibility function:

$$F(\mathbf{x}, y) = \phi(\mathbf{x}; \theta)^T W \psi(y), \tag{8.56}$$

where $\psi(y)$ is a semantic embedding of label y, and W is a learned transformation matrix. The model is trained to maximize the compatibility of correct pairs while minimizing that of incorrect pairs, often using a ranking loss:

$$\mathcal{L} = \sum_i \sum_{j \neq i} \max(0, \gamma - F(\mathbf{x}_i, y_i) + F(\mathbf{x}_i, y_j)), \tag{8.57}$$

where γ is a margin hyperparameter. Prototypical networks leverage metric-based meta-learning to learn a space where samples from the same class cluster together. Given a support set $S = \{(\mathbf{x}_i, y_i)\}$, the prototype for class c is computed as the mean embedding of its examples:

$$\mathbf{p}_c = \frac{1}{|S_c|} \sum_{\mathbf{x}_i \in S_c} \phi(\mathbf{x}_i; \theta). \tag{8.58}$$

A query sample $\mathbf{x}^*$ is classified by computing a distance metric, commonly the squared Euclidean distance:

$$p(y = c|\mathbf{x}^*) = \frac{\exp(-d(\phi(\mathbf{x}^*; \theta), \mathbf{p}_c))}{\sum_{c'} \exp(-d(\phi(\mathbf{x}^*; \theta), \mathbf{p}_{c'}))}. \tag{8.59}$$

Graph-based meta-learning frameworks construct a graph $G = (V, E)$ where nodes represent categories, and edges encode semantic or structural relationships. Given a graph convolutional network (GCN) with adjacency matrix A and feature matrix X, the node embeddings are computed iteratively as:

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}), \tag{8.60}$$

where $W^{(l)}$ are layer-specific learnable parameters and σ is a non-linearity. The final node embeddings serve as classifiers for unseen categories. Bayesian meta-learning introduces uncertainty quantification by modeling the parameters as distributions rather than point estimates. Let $\theta \sim p(\theta)$ be a prior distribution over parameters. The posterior is computed using Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{8.61}$$

Inference is performed by marginalizing over the posterior:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, \theta)p(\theta|\mathcal{D})d\theta. \tag{8.62}$$

Approximate inference methods such as variational Bayes or Monte Carlo sampling are used to estimate the integral. Each of these approaches forms the foundation of meta-learning in zero-shot learning, providing a mechanism to generalize across unseen categories using transferable knowledge.

8.2.1 Model-Agnostic Meta-Learning (MAML)

Model-Agnostic Meta-Learning (MAML) is a meta-learning algorithm designed to enable fast adaptation of a model to new tasks with minimal training data. The core idea behind MAML is to learn an initial set of parameters θ that serve as a good starting point for fine-tuning on new tasks using only a few gradient steps. This is particularly useful for zero-shot learning, where a model must generalize to unseen tasks without explicit training on them. Given a distribution over tasks $p(\mathcal{T})$, each task $\mathcal{T}_i$ consists of a dataset $\mathcal{D}_i = \{(x_{i,j}, y_{i,j})\}_{j=1}^{N}$. The goal is to find an initialization θ that minimizes the expected loss across all tasks after adaptation. Formally, the optimization problem can be written as

$$\min_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}}(U(\theta, \mathcal{T}))\right] \tag{8.63}$$

where $U(\theta, \mathcal{T})$ represents the model parameters after adaptation on task $\mathcal{T}$. In the standard supervised learning setting, given a task-specific loss function $\mathcal{L}_{\mathcal{T}}(\theta)$, the adaptation process involves performing one or more gradient descent steps with respect to this loss function. The adapted parameters for task $\mathcal{T}_i$ after one step of gradient descent are

$$\theta_i' = \theta - \alpha\nabla_{\theta}\mathcal{L}_{\mathcal{T}_i}(\theta) \tag{8.64}$$

where α is the learning rate. The meta-objective is then formulated as

$$\min_{\theta} \sum_{i} \mathcal{L}_{\mathcal{T}_i}(\theta_i') \tag{8.65}$$

which involves optimizing the initial parameters θ to ensure that the adapted parameters θ_i' yield low loss across tasks. This optimization is performed using gradient descent, leading to a second-order gradient update of the form

$$\theta \leftarrow \theta - \beta \sum_{i} \nabla_{\theta}\mathcal{L}_{\mathcal{T}_i}(\theta - \alpha\nabla_{\theta}\mathcal{L}_{\mathcal{T}_i}(\theta)) \tag{8.66}$$

where β is the meta-learning rate. The second-order term in this update makes MAML computationally expensive, but it allows for more expressive updates. In zero-shot learning, where the model must generalize to tasks without direct exposure to them during meta-training, the effectiveness of MAML is attributed to its ability to produce an initialization that is inherently generalizable. Specifically, if the task distribution $p(\mathcal{T})$ is designed such that the training tasks span a diverse range of possible learning scenarios, then the learned initialization can generalize to unseen tasks by leveraging a smooth loss landscape across task variations. Mathematically, this can be expressed as minimizing the expected task-generalization error

$$\min_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\mathbb{E}_{\mathcal{T}'\sim p_{\text{test}}(\mathcal{T}')} \mathcal{L}_{\mathcal{T}'}(U(\theta, \mathcal{T}))\right] \tag{8.67}$$

where $p_{\text{test}}(\mathcal{T}')$ represents the distribution over unseen tasks. To approximate this expectation, a common strategy is to incorporate a regularization term that promotes smooth adaptation across task variations, yielding the regularized meta-objective

$$\min_{\theta} \sum_{i} \mathcal{L}_{\mathcal{T}_i}(\theta_i') + \lambda R(\theta) \tag{8.68}$$

where $R(\theta)$ is a regularization function that penalizes large gradients or promotes parameter smoothness, and λ controls the strength of this regularization. The zero-shot capability of MAML can be further improved by explicitly optimizing for robustness in the presence of distribution shifts. This can be achieved by modifying the meta-objective to incorporate an adversarial worst-case term

$$\min_{\theta} \max_{\mathcal{T}'\sim p_{\text{adv}}(\mathcal{T}')} \mathcal{L}_{\mathcal{T}'}(U(\theta, \mathcal{T})) \tag{8.69}$$

where $p_{\text{adv}}(\mathcal{T}')$ represents an adversarially chosen distribution over test tasks. This leads to a minimax optimization problem, ensuring that the learned initialization is robust to extreme variations in task distributions. Additionally, when applied to function approximation problems such as few-shot regression or reinforcement learning, MAML can be interpreted as an implicit form of hierarchical Bayesian inference. In this interpretation, the meta-learned parameters θ serve as a prior, and task-specific fine-tuning corresponds to a Bayesian posterior update. This perspective connects MAML with variational inference methods, where the meta-learning process can be rewritten as optimizing the evidence lower bound (ELBO)

$$\max_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\log p_{\mathcal{T}}(\mathcal{D}|\theta) - D_{\text{KL}}(q_{\mathcal{T}}(\theta') \| p(\theta))\right] \tag{8.70}$$

where $q_{\mathcal{T}}(\theta')$ represents the posterior distribution over task-specific parameters and $p(\theta)$ is the learned meta-prior. The KL-divergence term D_{KL} enforces consistency between the meta-learned initialization and the task-specific updates.

8.2.2 Meta-embedding strategy

The meta-embedding strategy for zero-shot learning (ZSL) fundamentally relies on constructing an optimal embedding space where the semantic representations of seen and unseen classes align with feature representations derived from visual or textual modalities. Mathematically, given a set of seen class-label pairs $\{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ represents the feature vector extracted from an image (e.g., using a deep convolutional neural network), and $y_i \in \mathcal{Y}_{\text{seen}}$ is the corresponding class label, the objective is to learn a function $f : \mathbb{R}^d \to \mathbb{R}^m$ that maps feature representations to a semantic space where class relationships are preserved. The challenge is to generalize this function to a set of unseen class labels $\mathcal{Y}_{\text{unseen}}$, such that for an unseen sample x_u, its embedding $f(x_u)$ aligns closely with the corresponding semantic representation of its true class label.

To achieve this, a semantic embedding space $S \in \mathbb{R}^m$ is constructed, where each class c (both seen and unseen) is represented by a prototype vector $s_c \in \mathbb{R}^m$. The embeddings are often derived from textual descriptions (e.g., word vectors, attribute annotations) and must be aligned with the visual space. This alignment is achieved by learning a compatibility function $\mathcal{F} : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$ that quantifies the compatibility between an input feature vector x and a class prototype s_c. One common choice for $\mathcal{F}$ is a bilinear compatibility function:

$$\mathcal{F}(x, s_c) = x^T W s_c \tag{8.71}$$

where $W \in \mathbb{R}^{d\times m}$ is a learnable transformation matrix. The objective function for training is then given by a ranking loss that enforces the correct pair (x, s_y) to have a higher compatibility score than incorrect class pairs:

$$\sum_{i=1}^{N} \sum_{c\neq y_i} \max\left(0, \gamma - \mathcal{F}(x_i, s_{y_i}) + \mathcal{F}(x_i, s_c)\right) \tag{8.72}$$

where γ is a margin parameter ensuring separation between correct and incorrect matches. To enable generalization to unseen classes, meta-embedding strategies are introduced, where a meta-learned function $\mathcal{G}$ constructs a class prototype dynamically:

$$s_c = \mathcal{G}(z_c) \tag{8.73}$$

where z_c is a raw class descriptor (e.g., word embeddings, attribute vectors) and $\mathcal{G}$ is a neural network that transforms z_c into a semantically meaningful embedding. The training objective extends to minimizing the discrepancy between predicted and actual prototypes:

$$\min_{\mathcal{G}} \sum_{c\in\mathcal{Y}_{\text{seen}}} \|\mathcal{G}(z_c) - s_c\|^2 \tag{8.74}$$

so that for unseen classes, $s_c = \mathcal{G}(z_c)$ provides a meaningful representation. This ensures that when an unseen instance x_u is projected into the space, the classification decision:

$$\hat{y} = \arg \max_{c\in\mathcal{Y}_{\text{unseen}}} \mathcal{F}(x_u, s_c) \tag{8.75}$$

is based on learned semantic relationships rather than direct supervision. Additionally, variational methods can be employed to refine embeddings by incorporating uncertainty estimation via a probabilistic latent space:

$$p(s_c|z_c) = \mathcal{N}(\mu_{\mathcal{G}}(z_c), \Sigma_{\mathcal{G}}(z_c)) \tag{8.76}$$

where $\mu_{\mathcal{G}}(z_c)$ and $\Sigma_{\mathcal{G}}(z_c)$ are the mean and covariance functions modeled via neural networks. The overall loss function then includes a Kullback-Leibler (KL) divergence term to regularize the learned distributions:

$$\mathcal{L} = \sum_{c\in\mathcal{Y}_{\text{seen}}} \|s_c - \mu_{\mathcal{G}}(z_c)\|^2 + \lambda \sum_{c\in\mathcal{Y}_{\text{seen}}} D_{\text{KL}}\left(p(s_c|z_c)\|\mathcal{N}(0, I)\right) \tag{8.77}$$

where λ is a regularization parameter. By integrating these techniques, meta-embedding strategies provide a robust and scalable approach for zero-shot learning, ensuring that unseen class representations align meaningfully with visual features.

8.2.3 Metric-based meta-learning

Metric-based meta-learning for Zero-Shot Learning (ZSL) is fundamentally rooted in the concept of learning a function that embeds data points into a space where semantic similarities are preserved, facilitating knowledge transfer from seen to unseen classes. Let $\mathcal{X}$ denote the input space, $\mathcal{Y}$ the label space, and $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_s}$ be the training set, where $y_i \in \mathcal{Y}_s$, the set of seen class labels. The goal is to learn a function $f : \mathcal{X} \to \mathbb{R}^d$ that maps input instances into an embedding space where classification is performed based on distances to class prototypes, enabling generalization to an unseen class set $\mathcal{Y}_u$ without direct training data.

The embedding function $f(\mathbf{x})$ is parameterized as a deep neural network that learns a transformation from the raw input space to a metric space with a well-defined distance function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. One commonly employed distance function is the squared Euclidean distance:

$$d(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 \tag{8.78}$$

where $\mathbf{z}_i = f(\mathbf{x}_i)$ and $\mathbf{z}_j = f(\mathbf{x}_j)$. Alternatively, cosine similarity is often used for improved generalization:

$$d_{\cos}(\mathbf{z}_i, \mathbf{z}_j) = 1 - \frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\| \|\mathbf{z}_j\|} \tag{8.79}$$

A central component of metric-based meta-learning is the construction of prototypes $\mathbf{c}_y$ for each class, which serve as the representative embeddings for classification. The prototype of a class y is computed as the mean embedding of all support examples belonging to that class:

$$\mathbf{c}_y = \frac{1}{|\mathcal{S}_y|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}_y} f(\mathbf{x}_i) \tag{8.80}$$

where $\mathcal{S}_y$ is the set of all training instances labeled as y. Given a query instance $\mathbf{x}_q$, its class assignment is determined by computing the distance to all class prototypes and selecting the closest one:

$$\hat{y}_q = \arg\min_{y \in \mathcal{Y}_s} d(f(\mathbf{x}_q), \mathbf{c}_y) \tag{8.81}$$

In the meta-learning paradigm, training occurs episodically, where each episode simulates a ZSL scenario with a small support set $\mathcal{S}$ and a query set $\mathcal{Q}$. The meta-objective minimizes the classification loss over the query set using a nearest-prototype strategy:

$$\mathcal{L}_{\text{meta}} = \sum_{(\mathbf{x}_q, y_q) \in \mathcal{Q}} \ell(y_q, \hat{y}_q) \tag{8.82}$$

where ℓ is typically the cross-entropy loss:

$$\ell(y_q, \hat{y}_q) = -\log \frac{\exp(-d(f(\mathbf{x}_q), \mathbf{c}_{y_q}))}{\sum_{y' \in \mathcal{Y}_s} \exp(-d(f(\mathbf{x}_q), \mathbf{c}_{y'}))} \tag{8.83}$$

To enable zero-shot generalization, semantic descriptions of unseen classes, represented as attribute vectors $\mathbf{a}_y \in \mathbb{R}^m$, guide prototype construction. Instead of computing class prototypes directly from samples, they are inferred via a mapping function $g : \mathbb{R}^m \to \mathbb{R}^d$ that aligns semantic and visual spaces:

$$\mathbf{c}_y = g(\mathbf{a}_y) = W\mathbf{a}_y \tag{8.84}$$

where $W \in \mathbb{R}^{d \times m}$ is a learnable transformation matrix. The classification decision for an unseen query sample then follows:

$$\hat{y}_q = \arg\min_{y \in \mathcal{Y}_u} d(f(\mathbf{x}_q), g(\mathbf{a}_y)) \tag{8.85}$$

Training involves optimizing f and W jointly using episodic meta-learning, ensuring that f learns to generalize from seen to unseen classes by aligning feature embeddings with semantic representations. The meta-learning update follows:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{meta}}, \quad W \leftarrow W - \eta \nabla_W \mathcal{L}_{\text{meta}} \tag{8.86}$$

where θ denotes the parameters of the embedding function f, η is the learning rate, and gradients are computed over multiple episodes. The effectiveness of metric-based meta-learning in ZSL is enhanced by incorporating contrastive loss:

$$\mathcal{L}_{\text{contrastive}} = \sum_{i,j} y_{ij} d(\mathbf{z}_i, \mathbf{z}_j) + (1 - y_{ij}) \max(0, \tau - d(\mathbf{z}_i, \mathbf{z}_j)) \tag{8.87}$$

where y_{ij} is 1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same class, 0 otherwise, and τ is a margin parameter. This loss encourages similar instances to be closer while pushing dissimilar ones apart. Zero-shot generalization relies on the alignment between the semantic space and the learned metric space. Regularization techniques such as distribution calibration adjust the prototype computation by leveraging seen class distributions:

$$\mathbf{c}_y = \lambda g(\mathbf{a}_y) + (1-\lambda)\frac{1}{|\mathcal{N}_y|}\sum_{y' \in \mathcal{N}_y} \mathbf{c}_{y'} \tag{8.88}$$

where $\mathcal{N}_y$ is a set of semantically similar seen classes to y, and λ controls the balance between semantic projection and distribution calibration. By iteratively refining the metric space and semantic alignment, metric-based meta-learning enables effective zero-shot classification, where new classes are recognized based on learned semantic-visual correspondences without requiring additional samples. The integration of episodic training, prototype-based classification, contrastive learning, and semantic projection ensures that the learned model generalizes robustly to unseen categories.

8.2.4 Graph-based meta-learning

Graph-based meta-learning for zero-shot learning (ZSL) fundamentally relies on constructing a graph representation of semantic relationships between classes and leveraging meta-learning techniques to transfer knowledge from seen classes to unseen ones. Given a set of training classes $\mathcal{C}_s$ and a set of unseen classes $\mathcal{C}_u$ with $\mathcal{C}_s \cap \mathcal{C}_u = \emptyset$, the core idea is to embed both visual and semantic information into a structured graph $G = (V, E)$, where nodes V represent class prototypes and edges E encode relationships among them. The objective is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that can generalize to $\mathcal{C}_u$ despite lacking direct visual training examples.

Let each class $c \in \mathcal{C} = \mathcal{C}_s \cup \mathcal{C}_u$ be represented by a feature vector $\mathbf{z}_c \in \mathbb{R}^d$, typically derived from word embeddings such as Word2Vec or GloVe. The task is to learn a classifier $g(\mathbf{x}) = \arg\max_c P(c|\mathbf{x})$, where $P(c|\mathbf{x})$ is the probability of an input sample $\mathbf{x}$ belonging to class c. The underlying principle of the graph-based approach is to propagate information across the graph structure using message passing frameworks such as Graph Neural Networks (GNNs). The node feature update at layer t is given by

$$\mathbf{h}_c^{(t)} = \sigma\left(\sum_{c' \in \mathcal{N}(c)} w_{cc'} \mathbf{W}^{(t)} \mathbf{h}_{c'}^{(t-1)} + \mathbf{b}^{(t)}\right) \tag{8.89}$$

where $\mathbf{h}_c^{(t)}$ is the feature representation of class c at layer t, $\mathcal{N}(c)$ denotes the neighboring classes, $w_{cc'}$ is an edge weight encoding semantic similarity between class c and c', and σ is a nonlinear activation function such as ReLU. The parameters $\mathbf{W}^{(t)}$ and $\mathbf{b}^{(t)}$ are learned via backpropagation. To transfer knowledge, meta-learning is employed by training the model on a sequence of meta-tasks $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{Q}_i)$, where $\mathcal{S}_i$ is the support set and $\mathcal{Q}_i$ is the query set. The meta-learning objective is to minimize the expected loss across tasks:

$$\min_\theta \sum_i \mathbb{E}_{\mathcal{T}_i}\left[\mathcal{L}(g_\theta(\mathcal{S}_i), \mathcal{Q}_i)\right] \tag{8.90}$$

where g_θ is the classifier parameterized by θ and $\mathcal{L}$ is the classification loss. The classifier is optimized via an episodic training strategy, which mimics the zero-shot setting by holding out certain classes during training. Specifically, we define a parameterized distance metric $d_\phi(\mathbf{x}, \mathbf{h}_c)$ for classification, such as cosine similarity

$$d_\phi(\mathbf{x}, \mathbf{h}_c) = \frac{\mathbf{x}^T \mathbf{h}_c}{\|\mathbf{x}\|\|\mathbf{h}_c\|} \tag{8.91}$$

or a Mahalanobis distance

$$d_\phi(\mathbf{x}, \mathbf{h}_c) = (\mathbf{x} - \mathbf{h}_c)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{h}_c) \tag{8.92}$$

where $\mathbf{\Sigma}$ is a learned covariance matrix capturing intra-class variations. The final classification decision is made via a softmax layer

$$P(c|\mathbf{x}) = \frac{\exp(d_\phi(\mathbf{x}, \mathbf{h}_c))}{\sum_{c' \in \mathcal{C}} \exp(d_\phi(\mathbf{x}, \mathbf{h}_{c'}))} \tag{8.93}$$

To improve generalization to unseen classes, the model incorporates a regularization term based on the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the class graph. The regularization enforces smoothness in feature representations across the graph

$$\mathcal{L}_{\text{reg}} = \sum_{c,c'} w_{cc'} \|\mathbf{h}_c - \mathbf{h}_{c'}\|^2 \tag{8.94}$$

Thus, the final objective function combines classification loss and regularization

$$\mathcal{L} = \sum_i \mathbb{E}_{\mathcal{T}_i} \left[\mathcal{L}_{\text{class}}(g_\theta(\mathcal{S}_i), \mathcal{Q}_i)\right] + \lambda \mathcal{L}_{\text{reg}} \tag{8.95}$$

where λ is a trade-off parameter. To refine embeddings, contrastive learning can be employed by defining a contrastive loss

$$\mathcal{L}_{\text{contrast}} = - \sum_{(c,c^+)} \log \frac{\exp(d_\phi(\mathbf{h}_c, \mathbf{h}_{c^+}))}{\sum_{c' \in \mathcal{C}} \exp(d_\phi(\mathbf{h}_c, \mathbf{h}_{c'}))} \tag{8.96}$$

where (c, c^+) are semantically similar class pairs. The model is trained using stochastic gradient descent with updates

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L} \tag{8.97}$$

where η is the learning rate. The learned class representations are then used to classify unseen class samples by assigning them to the nearest prototype

$$c^* = \arg\max_c P(c|\mathbf{x}) \tag{8.98}$$

By leveraging the structured class graph, propagating knowledge through message passing, and optimizing via meta-learning, this approach enables robust zero-shot classification even when visual samples from unseen classes are unavailable.

8.2.5 Bayesian meta-learning

Bayesian meta-learning for zero-shot learning (ZSL) is fundamentally rooted in the probabilistic inference framework, where the goal is to infer the posterior distribution over tasks given a set of prior experiences. Let $\mathcal{T} = \{T_i\}_{i=1}^N$ denote a set of tasks, where each task T_i consists of a dataset $D_i = \{(\mathbf{x}_{ij}, y_{ij})\}_{j=1}^{m_i}$, with $\mathbf{x}_{ij} \in \mathbb{R}^d$ being the input feature and y_{ij} the corresponding label. The fundamental Bayesian formulation models the posterior over a hypothesis h, which is parametrized by a latent variable $\mathbf{z}$, as

$$p(h|D, \mathcal{T}) = \int p(h|\mathbf{z}) p(\mathbf{z}|D, \mathcal{T}) d\mathbf{z}. \tag{8.99}$$

In Bayesian meta-learning, we assume that the task distribution follows a hierarchical generative model. Given a hyperprior distribution $p(\mathbf{z})$, the task-specific parameters θ_i are sampled conditionally on $\mathbf{z}$ as

$$\theta_i \sim p(\theta|\mathbf{z}), \tag{8.100}$$

where θ_i are the task-specific parameters for T_i. The likelihood of observing dataset D_i given θ_i is then

$$p(D_i|\theta_i) = \prod_{j=1}^{m_i} p(y_{ij}|\mathbf{x}_{ij}, \theta_i). \tag{8.101}$$

For meta-learning, we define a posterior distribution over $\mathbf{z}$ conditioned on all observed tasks:

$$p(\mathbf{z}|\mathcal{T}) \propto p(\mathcal{T}|\mathbf{z})p(\mathbf{z}), \tag{8.102}$$

where

$$p(\mathcal{T}|\mathbf{z}) = \prod_{i=1}^{N} \int p(D_i|\theta_i)p(\theta_i|\mathbf{z})d\theta_i. \tag{8.103}$$

For zero-shot learning, where a new task T_* with dataset D_* is encountered without labeled examples, the goal is to infer the posterior predictive distribution

$$p(y_*|\mathbf{x}_*, \mathcal{T}) = \int p(y_*|\mathbf{x}_*, \theta_*)p(\theta_*|\mathcal{T})d\theta_*. \tag{8.104}$$

A variational Bayesian approach approximates $p(\mathbf{z}|\mathcal{T})$ using a variational distribution $q(\mathbf{z})$, typically assumed to be Gaussian:

$$q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mu, \Sigma). \tag{8.105}$$

The variational objective is to minimize the KL divergence

$$\mathcal{L} = D_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z}|\mathcal{T})), \tag{8.106}$$

which expands to the evidence lower bound (ELBO)

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{z})}\left[\sum_{i=1}^{N} \log p(D_i|\mathbf{z})\right] - D_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z})). \tag{8.107}$$

For amortized inference, we introduce an encoder $q(\mathbf{z}|D)$, parameterized as a neural network, such that

$$q(\mathbf{z}|D) = \mathcal{N}(\mu(D), \Sigma(D)). \tag{8.108}$$

During inference, for a new task T_*, the posterior predictive distribution is given by marginalizing out $\mathbf{z}$:

$$p(y_*|\mathbf{x}_*, \mathcal{T}) = \int p(y_*|\mathbf{x}_*, \mathbf{z})q(\mathbf{z}|\mathcal{T})d\mathbf{z}. \tag{8.109}$$

Using Monte Carlo approximation, the integral can be estimated as

$$p(y_*|\mathbf{x}_*, \mathcal{T}) \approx \frac{1}{K}\sum_{k=1}^{K} p(y_*|\mathbf{x}_*, \mathbf{z}_k), \tag{8.110}$$

where $\mathbf{z}_k \sim q(\mathbf{z}|\mathcal{T})$. The function $p(y_*|\mathbf{x}_*, \mathbf{z})$ is modeled using a deep neural network $f_\theta(\mathbf{x}_*, \mathbf{z})$, which outputs a probability distribution over classes. For zero-shot learning, the classifier must generalize to unseen classes by leveraging the inferred latent structure. This is achieved by incorporating semantic embeddings $\mathbf{e}(y)$ that define a probability distribution over labels:

$$p(y|\mathbf{x}, \mathbf{z}) \propto \exp(\mathbf{e}(y)^T f_\theta(\mathbf{x}, \mathbf{z})). \tag{8.111}$$

This formulation enables the model to predict classes that were never observed in the training set by mapping the latent variable $\mathbf{z}$ to a space shared across seen and unseen classes. The final optimization objective for meta-learning is

$$\mathcal{L}_{\text{meta}} = \sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{z}|D_i)}\left[\log p(D_i|\mathbf{z})\right] - D_{\mathrm{KL}}(q(\mathbf{z}|D_i)\|p(\mathbf{z})). \tag{8.112}$$

Through this Bayesian hierarchical approach, zero-shot learning emerges as a byproduct of inferring a shared structure across tasks, allowing for principled generalization to unseen distributions.

9 Neural Network Basics

Literature Review: Goodfellow et. al. (2016) [112] wrote one of the most comprehensive books on deep learning, covering the theoretical foundations of neural networks, optimization techniques, and probabilistic models. It is widely used in academic courses and research. Haykin (2009) [113] explained neural networks from a signal processing perspective, covering perceptrons, backpropagation, and recurrent networks with a strong mathematical approach. Schmidhuber (2015) [114] gave a historical and theoretical review of deep learning architectures, including recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory (LSTM). Bishop (2006) [115] gave a Bayesian perspective on neural networks and probabilistic graphical models, emphasizing the statistical underpinnings of learning. Poggio and Smale (2003) [116] established theoretical connections between neural networks, kernel methods, and function approximation. LeCun (2015) [117] discusses the principles behind modern deep learning, including backpropagation, unsupervised learning, and hierarchical feature extraction. Cybenko (1989) [58] proved the universal approximation theorem, demonstrating that a neural network with a single hidden layer can approximate any continuous function. Hornik et. al. (1989) [57] extended Cybenko's theorem, proving that multilayer perceptrons (MLPs) are universal function approximators. Pinkus (1999) [60] gave a rigorous mathematical discussion on neural networks from the perspective of approximation theory. Tishby and Zaslavsky (2015) [118] introduced the information bottleneck framework for understanding deep neural networks, explaining how networks learn to compress and encode information efficiently.

9.1 Perceptrons and Artificial Neurons

The perceptron is the simplest form of an artificial neural network, operating as a binary classifier. It computes the linear combination z of the input features $\vec{x} = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n$ and a corresponding weight vector $\vec{w} = [w_1, w_2, \ldots, w_n]^T \in \mathbb{R}^n$, augmented by a bias term $b \in \mathbb{R}$. This can be expressed as

$$z = \vec{w}^T\vec{x} + b = \sum_{i=1}^{n} w_i x_i + b. \tag{9.1}$$

To determine the output, this value is passed through the step activation function, defined mathematically as

$$\phi(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \tag{9.2}$$

Thus, the perceptron's decision-making process can be expressed as

$$y = \phi(\vec{w}^T\vec{x} + b), \tag{9.3}$$

where $y \in \{0, 1\}$. The equation $\vec{w}^T\vec{x} + b = 0$ defines a hyperplane in $\mathbb{R}^n$, which acts as the decision boundary. For any input $\vec{x}$, the classification is determined by the sign of $\vec{w}^T\vec{x}+b$, specifically $y = 1$ if $\vec{w}^T\vec{x}+b \geq 0$ and $y = 0$ otherwise. Geometrically, this classification corresponds to partitioning the input space into two distinct half-spaces. To train the perceptron, a supervised learning algorithm adjusts the weights $\vec{w}$ and the bias b iteratively using labeled training data $\{(\vec{x}_i, y_i)\}_{i=1}^m$, where y_i represents the ground truth. When the predicted output $y_{\text{pred}} = \phi(\vec{w}^T\vec{x}_i + b)$ differs from y_i, the weight vector and bias are updated according to the rule

$$\vec{w} \leftarrow \vec{w} + \eta(y_i - y_{\text{pred}})\vec{x}_i, \tag{9.4}$$

and

$$b \leftarrow b + \eta(y_i - y_{\text{pred}}), \tag{9.5}$$

where $\eta > 0$ is the learning rate. Each individual weight w_j is updated as

$$w_j \leftarrow w_j + \eta(y_i - y_{\text{pred}})x_{ij}. \tag{9.6}$$

For a linearly separable dataset, the Perceptron Convergence Theorem asserts that the algorithm will converge to a solution after a finite number of updates. Specifically, the number of updates is bounded by

$$\frac{R^2}{\gamma^2}, \tag{9.7}$$

where $R = \max_i \|\vec{x}_i\|$ is the maximum norm of the input vectors, and γ is the minimum margin, defined as

$$\gamma = \min_i \frac{y_i(\vec{w}^T\vec{x}_i + b)}{\|\vec{w}\|}. \tag{9.8}$$

The limitations of the perceptron, particularly its inability to solve linearly inseparable problems such as the XOR problem, necessitate the extension to artificial neurons with non-linear activation functions. A popular choice is the sigmoid activation function

$$\phi(z) = \frac{1}{1 + e^{-z}}, \tag{9.9}$$

which maps $z \in \mathbb{R}$ to the continuous interval $(0, 1)$. The derivative of the sigmoid function, essential for gradient-based optimization, is

$$\phi'(z) = \phi(z)(1 - \phi(z)). \tag{9.10}$$

Another widely used activation function is the hyperbolic tangent $\tanh(z)$, defined as

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{9.11}$$

with derivative

$$\tanh'(z) = 1 - \tanh^2(z). \tag{9.12}$$

ReLU, or Rectified Linear Unit, is defined as

$$\phi(z) = \max(0, z), \tag{9.13}$$

with derivative

$$\phi'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0. \end{cases} \tag{9.14}$$

These non-linear activations enable the network to approximate non-linear decision boundaries, a capability absent in the perceptron. Artificial neurons form the building blocks of multi-layer perceptrons (MLPs), where neurons are organized into layers. For an L-layer network, the input $\vec{x}$ is transformed layer by layer. At layer l, the output is

$$\vec{z}^{(l)} = \phi^{(l)}(\vec{W}^{(l)}\vec{z}^{(l-1)} + \vec{b}^{(l)}), \tag{9.15}$$

where $\vec{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix, $\vec{b}^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector, and $\phi^{(l)}$ is the activation function. The network's output is

$$\hat{\vec{y}} = \phi^{(L)}(\vec{W}^{(L)}\vec{z}^{(L-1)} + \vec{b}^{(L)}). \tag{9.16}$$

The Universal Approximation Theorem guarantees that MLPs with sufficient neurons and nonlinear activations can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ to arbitrary precision. Formally, for any $\epsilon > 0$, there exists an MLP $g(\vec{x})$ such that

$$\|f(\vec{x}) - g(\vec{x})\|_\infty < \epsilon \tag{9.17}$$

for all $\vec{x} \in \mathbb{R}^n$. Training an MLP minimizes a loss function L that quantifies the error between predicted outputs $\hat{\vec{y}}$ and ground truth labels $\vec{y}$. For regression, the mean squared error is

$$L = \frac{1}{m}\sum_{i=1}^{m} \|\hat{\vec{y}}_i - \vec{y}_i\|^2, \tag{9.18}$$

and for classification, the cross-entropy loss is

$$L = -\frac{1}{m}\sum_{i=1}^{m} \left[\vec{y}_i^T \log \hat{\vec{y}}_i + (1 - \vec{y}_i)^T \log(1 - \hat{\vec{y}}_i)\right]. \tag{9.19}$$

Optimization uses stochastic gradient descent (SGD), updating parameters $\Theta = \{\vec{W}^{(l)}, \vec{b}^{(l)}\}_{l=1}^L$ as

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta L. \tag{9.20}$$

Gradients are computed via backpropagation:

$$\frac{\partial L}{\partial \vec{W}^{(l)}} = \delta^{(l)} \vec{z}^{(l-1)T}, \tag{9.21}$$

where $\delta^{(l)}$ is the error signal at layer l, recursively calculated as

$$\delta^{(l)} = (\vec{W}^{(l+1)T}\delta^{(l+1)}) \circ \phi'^{(l)}(\vec{z}^{(l)}). \tag{9.22}$$

This recursive structure, combined with chain rule applications, efficiently propagates error signals from the output layer back to the input layer.

Artificial neurons and their extensions have thus provided the foundation for modern deep learning. Their mathematical underpinnings and computational frameworks are instrumental in solving a wide array of problems, from classification and regression to complex decision-making. The interplay of linear algebra, calculus, and optimization theory in their formulation ensures that these networks are both theoretically robust and practically powerful.

9.2 Feedforward Neural Networks

Feedforward neural networks (FNNs) are mathematical constructs designed to approximate arbitrary mappings $f : \mathbb{R}^n \to \mathbb{R}^m$ by composing affine transformations and nonlinear activation functions. At their core, these networks consist of L layers, where each layer k transforms its input $\vec{a}_{k-1} \in \mathbb{R}^{m_{k-1}}$ into an output $\vec{a}_k \in \mathbb{R}^{m_k}$ via the operation

$$\vec{a}_k = f_k(W_k \vec{a}_{k-1} + \vec{b}_k). \tag{9.23}$$

Here, $W_k \in \mathbb{R}^{m_k \times m_{k-1}}$ represents the weight matrix, $\vec{b}_k \in \mathbb{R}^{m_k}$ is the bias vector, and $f_k : \mathbb{R}^{m_k} \to \mathbb{R}^{m_k}$ is a component-wise activation function. Formally, if we denote the input layer as $\vec{a}_0 = \vec{x}$, the final output of the network, $\vec{y} \in \mathbb{R}^m$, is given by $\vec{a}_L = f_L(W_L\vec{a}_{L-1} + \vec{b}_L)$. Each transformation in this sequence can be described as $\vec{z}_k = W_k\vec{a}_{k-1} + \vec{b}_k$, followed by the activation $\vec{a}_k = f_k(\vec{z}_k)$. The affine transformation $\vec{z}_k = W_k\vec{a}_{k-1} + \vec{b}_k$ encapsulates the linear combination of inputs with weights W_k and the addition of biases $\vec{b}_k$. For any two layers k and $k+1$, the overall transformation can be represented by

$$\vec{z}_{k+1} = W_{k+1}(W_k\vec{a}_{k-1} + \vec{b}_k) + \vec{b}_{k+1}. \tag{9.24}$$

Expanding this, we have

$$\vec{z}_{k+1} = W_{k+1}W_k\vec{a}_{k-1} + W_{k+1}\vec{b}_k + \vec{b}_{k+1}. \tag{9.25}$$

Without the nonlinearity introduced by f_k, the network reduces to a single affine transformation

$$\vec{y} = W\vec{x} + \vec{b}, \tag{9.26}$$

where $W = W_L W_{L-1} \cdots W_1$ and

$$\vec{b} = W_L W_{L-1} \cdots W_2\vec{b}_1 + \cdots + \vec{b}_L. \tag{9.27}$$

Thus, the incorporation of nonlinear activation functions is critical, as it enables the network to approximate non-linear mappings. Activation functions f_k are applied element-wise to the pre-activation vector $\vec{z}_k$. The choice of activation significantly affects the network's behavior and training. For example, the sigmoid activation $f(x) = \frac{1}{1+e^{-x}}$ compresses inputs into the range $(0, 1)$ and has a derivative given by

$$f'(x) = f(x)(1 - f(x)). \tag{9.28}$$

The hyperbolic tangent activation $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ maps inputs to $(-1, 1)$ with a derivative

$$f'(x) = 1 - \tanh^2(x). \tag{9.29}$$

The ReLU activation $f(x) = \max(0, x)$, commonly used in modern networks, has a derivative

$$f'(x) = \begin{cases} 1 & x > 0, \\ 0 & x \leq 0. \end{cases} \tag{9.30}$$

These derivatives are essential for gradient-based optimization. The objective of training a feedforward neural network is to minimize a loss function $\mathcal{L}$, which measures the discrepancy between the predicted outputs $\vec{y}_i$ and the true targets $\vec{t}_i$ over a dataset $\{(\vec{x}_i, \vec{t}_i)\}_{i=1}^N$. For regression problems, the mean squared error (MSE) is often used, given by

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\vec{y}_i - \vec{t}_i\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m}(y_{i,j} - t_{i,j})^2. \tag{9.31}$$

In classification tasks, the cross-entropy loss is widely employed, defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} t_{i,j}\log(y_{i,j}), \tag{9.32}$$

where $t_{i,j}$ represents the one-hot encoded labels. The gradient of $\mathcal{L}$ with respect to the network parameters is computed using backpropagation, which applies the chain rule iteratively to propagate errors from the output layer to the input layer. During backpropagation, the error signal at the output layer is computed as

$$\delta_L = \frac{\partial \mathcal{L}}{\partial \vec{z}_L} = \nabla_{\vec{y}}\mathcal{L} \odot f_L'(\vec{z}_L), \tag{9.33}$$

where $\odot$ denotes the Hadamard product. For hidden layers, the error signal propagates backward as

$$\delta_k = (W_{k+1}^T \delta_{k+1}) \odot f_k'(\vec{z}_k). \tag{9.34}$$

The gradients of the loss with respect to the weights and biases are then given by

$$\frac{\partial \mathcal{L}}{\partial W_k} = \delta_k \vec{a}_{k-1}^T, \quad \frac{\partial \mathcal{L}}{\partial \vec{b}_k} = \delta_k. \tag{9.35}$$

These gradients are used to update the parameters through optimization algorithms like stochastic gradient descent (SGD), where

$$W_k \leftarrow W_k - \eta \frac{\partial \mathcal{L}}{\partial W_k}, \quad \vec{b}_k \leftarrow \vec{b}_k - \eta \frac{\partial \mathcal{L}}{\partial \vec{b}_k}, \tag{9.36}$$

with $\eta > 0$ as the learning rate. The universal approximation theorem rigorously establishes that a feedforward neural network with at least one hidden layer and sufficiently many neurons can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ on a compact domain $D \subset \mathbb{R}^n$. Specifically, for any $\epsilon > 0$, there exists a network $\hat{f}$ such that $\|f(\vec{x}) - \hat{f}(\vec{x})\| < \epsilon$ for all $\vec{x} \in D$. This expressive capability arises because the composition of affine transformations and nonlinear activations allows the network to approximate highly complex functions by partitioning the input space into regions and assigning different functional behaviors to each.

In summary, feedforward neural networks are a powerful mathematical framework grounded in linear algebra, calculus, and optimization. Their capacity to model intricate mappings between input and output spaces has made them a cornerstone of machine learning, with rigorous mathematical principles underpinning their structure and training. The combination of affine transformations, nonlinear activations, and gradient-based optimization enables these networks to achieve unparalleled flexibility and performance in a wide range of applications.

9.3 Activation Functions

In the context of **neural networks**, activation functions serve as an essential component that enables the network to approximate complex, non-linear mappings. When a neural network processes input data, each neuron computes a weighted sum of the inputs and then applies an activation function $\sigma(z)$ to produce the output. Mathematically, let the input vector to the neuron be $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and let the weight vector associated with these inputs be $\mathbf{w} = (w_1, w_2, \ldots, w_n)$. The corresponding bias term is denoted as b. The net input z to the activation function is then given by:

$$z = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b \tag{9.37}$$

where $\mathbf{w}^\top \mathbf{x}$ represents the dot product of the weight vector and the input vector. The activation function $\sigma(z)$ is then applied to this net input to obtain the output of the neuron a:

$$a = \sigma(z) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{9.38}$$

The activation function introduces a *non-linearity* into the neuron's response, which is a crucial aspect of neural networks because, without it, the network would only be able to perform linear transformations of the input data, limiting its ability to approximate complex, real-world functions. The non-linearity introduced by $\sigma(z)$ is fundamental because it enables the network to capture intricate relationships between the input and output, making neural networks capable of solving problems that require hierarchical feature extraction, such as image classification, time-series forecasting, and language modeling. The importance of non-linearity is most clearly evident when considering the mathematical formulation of a multi-layer neural network. For a feed-forward neural network with L layers, the output $\hat{y}$ of the network is given by the composition of successive affine transformations and activation functions. Let $\mathbf{x}$ denote the input vector, W_k and b_k be the weight matrix and bias vector for the k-th layer, and σ_k be the activation function for the k-th layer. The output of the network is:

$$\hat{y} = \sigma_L(W_L \sigma_{L-1}(W_{L-1} \dots \sigma_1(W_1 \mathbf{x} + b_1) + b_2) + \cdots + b_L). \tag{9.39}$$

If $\sigma(z)$ were a linear function, say $\sigma(z) = c \cdot z$ for some constant c, the composition of such functions would still result in a linear function. Specifically, if each σ_k were linear, the overall network function would simplify to a single linear transformation:

$$\hat{y} = c_1 \cdot \mathbf{x} + c_2, \tag{9.40}$$

where c_1 and c_2 are constants dependent on the parameters of the network. In this case, the network would have no greater expressive power than a simple linear regression model, regardless of the number of layers. Thus, the *non-linearity* introduced by activation functions allows neural networks to approximate *any continuous function*, as guaranteed by the *universal approximation theorem*. This theorem states that a feed-forward neural network with at least one hidden layer and a sufficiently large number of neurons can approximate any continuous function $f(\mathbf{x})$, provided the activation function is non-linear and the network has enough capacity.

Next, consider the mathematical properties that the activation function $\sigma(z)$ must possess. First, it must be *differentiable* to allow the use of gradient-based optimization methods like *backpropagation* for training. Backpropagation relies on the *chain rule* of calculus to compute the gradients of the loss function $\mathcal{L}$ with respect to the parameters (weights and biases) of the network. Suppose $\mathcal{L} = \mathcal{L}(\hat{y}, \mathbf{y})$ is the loss function, where $\hat{y}$ is the predicted output of the network and $\mathbf{y}$ is the true label. During training, we compute the gradient of $\mathcal{L}$ with respect to the weights using the chain rule. Let $a_k = \sigma_k(z_k)$ represent the output of the activation function at layer k, where z_k is the input to the activation function. The gradient of the loss with respect to the weights at layer k is given by:

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial W_k}. \tag{9.41}$$

The term $\frac{\partial a_k}{\partial z_k}$ is the *derivative* of the activation function, which must exist and be well-defined for gradient-based optimization to work effectively. If the activation function is not differentiable, the backpropagation algorithm cannot compute the gradients, preventing the training process from proceeding.

Now consider the specific forms of activation functions commonly used in practice. The **sigmoid** activation function is one of the most well-known, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{9.42}$$

Its derivative is:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)), \tag{9.43}$$

which can be derived by applying the chain rule to the expression for $\sigma(z)$. Although sigmoid is differentiable and smooth, it suffers from the *vanishing gradient problem*, especially for large positive or negative values of z. Specifically, as $z \to \infty$, $\sigma'(z) \to 0$, and similarly as $z \to -\infty$, $\sigma'(z) \to 0$. This results in very small gradients during backpropagation, making it difficult for the network to learn when the input values become extreme. To mitigate the vanishing gradient problem, the **hyperbolic tangent (tanh)** function is often used as an alternative. It is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{9.44}$$

with derivative:

$$\tanh'(z) = 1 - \tanh^2(z). \tag{9.45}$$

The tanh function outputs values in the range $(-1, 1)$, which helps to center the data around zero. While the tanh function overcomes some of the vanishing gradient issues associated with the sigmoid function, it still suffers from the problem for large $|z|$, where the gradients approach zero. The **Rectified Linear Unit (ReLU)** is another commonly used activation function. It is defined as:

$$\text{ReLU}(z) = \max(0, z), \tag{9.46}$$

with derivative:

$$\text{ReLU}'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0. \end{cases} \tag{9.47}$$

ReLU is particularly advantageous because it is computationally efficient, as it only requires a comparison to zero. Moreover, for positive values of z, the derivative is constant and equal to 1, which helps avoid the vanishing gradient problem. However, ReLU can suffer from the *dying ReLU problem*, where neurons output zero for all inputs if the weights are initialized poorly or if the learning rate is too high, leading to inactive neurons that do not contribute to the learning process. To address the dying ReLU problem, the **Leaky ReLU** activation function is introduced, defined as:

$$\text{Leaky ReLU}(z) = \begin{cases} z, & z > 0, \\ \alpha z, & z \leq 0, \end{cases} \tag{9.48}$$

where α is a small constant, typically chosen to be 0.01. The derivative of the Leaky ReLU is:

$$\text{Leaky ReLU}'(z) = \begin{cases} 1, & z > 0, \\ \alpha, & z \leq 0. \end{cases} \tag{9.49}$$

Leaky ReLU ensures that neurons do not become entirely inactive by allowing a small, non-zero gradient for negative values of z. Finally, for classification tasks, particularly when there are multiple classes, the **Softmax** activation function is often used in the output layer of the neural network. The Softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{9.50}$$

where z_i is the input to the i-th neuron in the output layer, and the denominator ensures that the outputs sum to 1, making them interpretable as probabilities. The Softmax function is typically used in multi-class classification problems, where the network must predict one class out of several possible categories.

In summary, activation functions are a vital component of neural networks, enabling them to learn intricate patterns in data, allowing for the successful application of neural networks to diverse tasks. Different activation functions—such as sigmoid, tanh, ReLU, Leaky ReLU, and Softmax—each offer distinct advantages and limitations, and their choice significantly impacts the performance and training dynamics of the neural network.

9.4 Loss Functions

In neural networks, the **loss function** is a crucial mathematical tool that quantifies the difference between the predicted output of the model and the true output or target. Let $\mathbf{x}_i$ be the input vector and $\mathbf{y}_i$ the corresponding target vector for the i-th training example. The network, parameterized by weights $\mathbf{W}$, generates a prediction denoted as $\hat{\mathbf{y}}_i = f(\mathbf{x}_i; \mathbf{W})$, where $f(\mathbf{x}_i; \mathbf{W})$ represents the model's output. The objective of training the neural network is to minimize the discrepancy between the predicted output $\hat{\mathbf{y}}_i$ and the true label $\mathbf{y}_i$ across all training examples, effectively learning the mapping function from inputs to outputs. A typical objective function is the **average loss** over a dataset of N samples:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{9.51}$$

where $L(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ represents the loss function that computes the error between the true output $\mathbf{y}_i$ and the predicted output $\hat{\mathbf{y}}_i$ for each data point. To minimize this objective function, optimization algorithms such as **gradient descent** are used. The general update rule for the weights $\mathbf{W}$ is given by:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \tag{9.52}$$

where η is the **learning rate**, and $\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$ is the gradient of the loss function with respect to the weights. The gradient is computed using **backpropagation**, which applies the **chain rule** of calculus to propagate the error backward through the network, updating the parameters to minimize the loss. For this, we use the partial derivatives of the loss with respect to each layer's weights and biases, ensuring the error is distributed appropriately across all layers. For regression tasks, the **Mean Squared Error (MSE)** loss is frequently used. This loss function quantifies the error as the average squared difference between the predicted and true values. The MSE for a dataset of N examples is given by:

$$L_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{9.53}$$

where $\hat{\mathbf{y}}_i = f(\mathbf{x}_i; \mathbf{W})$ is the network's predicted output for the i-th input $\mathbf{x}_i$. The gradient of the MSE with respect to the network's output $\hat{\mathbf{y}}_i$ is:

$$\frac{\partial L_{\text{MSE}}}{\partial \hat{\mathbf{y}}_i} = 2\,(\hat{\mathbf{y}}_i - \mathbf{y}_i) \tag{9.54}$$

This gradient guides the weight update in the direction that minimizes the squared error, leading to a better fit of the model to the training data. For **classification tasks**, the **cross-entropy loss** is often employed, as it is particularly well-suited to tasks where the output is a probability distribution over multiple classes. In the binary classification case, where the target label $\mathbf{y}_i$ is either 0 or 1, the binary cross-entropy loss function is defined as:

$$L_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{9.55}$$

where $\hat{y}_i = f(\mathbf{x}_i; \mathbf{W})$ is the predicted probability that the i-th sample belongs to the positive class (i.e., class 1). For multiclass classification, where the target label $\mathbf{y}_i$ is a one-hot encoded vector representing the true class, the general form of the cross-entropy loss is:

$$L_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \tag{9.56}$$

where C is the number of classes, and $\hat{y}_{i,c} = f(\mathbf{x}_i; \mathbf{W})$ is the predicted probability that the i-th sample belongs to class c. The gradient of the cross-entropy loss with respect to the predicted probabilities $\hat{\mathbf{y}}_i$ is:

$$\frac{\partial L_{\text{CE}}}{\partial \hat{y}_{i,c}} = \hat{y}_{i,c} - y_{i,c} \tag{9.57}$$

This gradient facilitates the weight update by adjusting the model's parameters to reduce the difference between the predicted probabilities and the actual class labels.

In neural network training, the optimization process often involves regularization techniques to prevent **overfitting**, especially in cases with high-dimensional data or deep networks. **L2 regularization** (also known as **Ridge regression**) is one common approach, which penalizes large weights by adding a term proportional to the squared L2 norm of the weights to the loss function. The regularized loss function becomes:

$$L_{\text{reg}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{n} W_j^2 \tag{9.58}$$

where λ is the regularization strength, and W_j represents the parameters of the network. The gradient of the regularized loss with respect to the weights is:

$$\frac{\partial L_{\text{reg}}}{\partial W_j} = \frac{\partial L_{\text{MSE}}}{\partial W_j} + 2\lambda W_j \tag{9.59}$$

This additional term discourages large values of the weights, reducing the complexity of the model and helping it generalize better to unseen data. Another form of regularization is **L1 regularization** (or **Lasso regression**), which promotes sparsity in the model by adding the L1 norm of the weights to the loss function. The L1 regularized loss function is:

$$L_{\text{reg}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{n} |W_j| \tag{9.60}$$

The gradient of this regularized loss function with respect to the weights is:

$$\frac{\partial L_{\text{reg}}}{\partial W_j} = \frac{\partial L_{\text{MSE}}}{\partial W_j} + \lambda \operatorname{sign}(W_j) \tag{9.61}$$

where $\text{sign}(W_j)$ is the sign function, which returns 1 for positive values of W_j, -1 for negative values, and 0 for $W_j = 0$. L1 regularization encourages the model to select only a small subset of features by forcing many of the weights to exactly zero, thus simplifying the model and promoting interpretability. The optimization process for neural networks can be viewed as solving a **non-convex optimization problem**, given the highly non-linear activation functions and the deep architectures typically used. In this context, **stochastic gradient descent (SGD)** is commonly employed to perform the optimization by updating the weights based on the gradient computed from a random mini-batch of the data. The update rule for SGD can be expressed as:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L_{\text{batch}} \tag{9.62}$$

where $\nabla_{\mathbf{W}} L_{\text{batch}}$ is the gradient of the loss function computed over the mini-batch, and η is the learning rate. Due to the non-convexity of the objective function, SGD tends to converge to a **local minimum** or a **saddle point**, rather than the global minimum, especially in deep neural networks with many layers.

In summary, the loss function plays a central role in guiding the optimization process in neural network training by quantifying the error between the predicted and true outputs. Different loss functions are employed depending on the nature of the problem, with MSE being common for regression and cross-entropy used for classification. Regularization techniques such as L2 and L1 regularization are incorporated to prevent overfitting and ensure better generalization. Through optimization algorithms like gradient descent, the neural network parameters are iteratively updated based on the gradients of the loss function, with the ultimate goal of minimizing the loss across all training examples.

10 Few-Shot Learning

Few-shot learning (FSL) refers to the problem of training a model to generalize to new tasks using only a limited number of labeled examples. Mathematically, given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ consisting of N labeled instances, a conventional supervised learning model aims to learn a mapping $f : \mathcal{X} \to \mathcal{Y}$ by minimizing an empirical risk functional of the form

$$\mathcal{L}(f) = \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) \tag{10.1}$$

where $\ell(\cdot, \cdot)$ is an appropriate loss function such as the cross-entropy loss

$$\ell(f(\mathbf{x}_i), y_i) = -\sum_{k=1}^{K} \nVdash(y_i = k) \log P(y_i = k|\mathbf{x}_i, \theta) \tag{10.2}$$

where θ denotes the model parameters. However, in a few-shot setting, the number of labeled examples per class is very small, typically denoted as K, where $K \ll N$, making it challenging to generalize in a conventional learning paradigm. Few-shot learning is often formalized using meta-learning approaches where the goal is to optimize a learning algorithm itself such that it can quickly adapt to new tasks using very few labeled samples. A common meta-learning formulation involves the optimization of a meta-objective defined over a distribution of tasks $\mathcal{T}$. Given a distribution $p(\mathcal{T})$, a meta-learner optimizes parameters θ such that the expected loss over tasks is minimized:

$$\min_{\theta} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}}(f_\theta) \right] \tag{10.3}$$

where each task $\mathcal{T}$ consists of a small training set $\mathcal{S}_{\text{train}}$ (the support set) and a validation set $\mathcal{S}_{\text{test}}$ (the query set). The loss function for a given task $\mathcal{T}$ is typically written as:

$$\mathcal{L}_{\mathcal{T}}(f_\theta) = \sum_{(\mathbf{x},y) \in \mathcal{S}_{\text{test}}} \ell(f_\theta^{\mathcal{S}_{\text{train}}}(\mathbf{x}), y) \tag{10.4}$$

where $f_\theta^{\mathcal{S}_{\text{train}}}$ denotes the model adapted using the small support set $\mathcal{S}_{\text{train}}$. A well-known approach to FSL is Model-Agnostic Meta-Learning (MAML), where the optimization is performed by updating θ such that it rapidly adapts to new tasks via a few gradient updates. Specifically, MAML defines a bi-level optimization problem where an inner loop computes task-specific parameters θ' via gradient descent:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}}(f_\theta) \tag{10.5}$$

where α is the step size. The outer loop then updates θ based on the loss incurred on the query set:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T} \sim p(\mathcal{T})} \nabla_\theta \mathcal{L}_{\mathcal{T}}(f_{\theta'}) \tag{10.6}$$

where β is another learning rate. The goal of MAML is to find an initialization θ such that a small number of gradient steps suffices for good generalization. Another prominent FSL approach is metric-based learning, where models learn an embedding function $g_\theta : \mathcal{X} \to \mathbb{R}^d$ such that a similarity function $S(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j))$ enables effective classification in a low-data regime. A widely used method is prototypical networks, which represent each class in the support set with a prototype:

$$\mathbf{c}_k = \frac{1}{|\mathcal{S}_{\text{train},k}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}_{\text{train},k}} g_\theta(\mathbf{x}_i) \tag{10.7}$$

where $\mathcal{S}_{\text{train},k}$ is the subset of the support set corresponding to class k. Classification is then performed using a softmax function over distances:

$$P(y = k|\mathbf{x}) = \frac{\exp(-d(g_\theta(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(g_\theta(\mathbf{x}), \mathbf{c}_{k'}))} \tag{10.8}$$

where $d(\cdot, \cdot)$ is typically the squared Euclidean distance. The training objective is the negative log-likelihood:

$$\mathcal{L} = - \sum_{(\mathbf{x}, y) \in \mathcal{S}_{\text{test}}} \log P(y|\mathbf{x}) \tag{10.9}$$

Bayesian methods in FSL adopt a probabilistic framework where uncertainty in model parameters is explicitly modeled using a prior $p(\theta)$. Given a support set $\mathcal{S}_{\text{train}}$, a Bayesian model infers a posterior over parameters:

$$p(\theta|\mathcal{S}_{\text{train}}) \propto p(\mathcal{S}_{\text{train}}|\theta) p(\theta) \tag{10.10}$$

A predictive distribution over a new query point $\mathbf{x}^*$ is obtained by marginalizing over θ:

$$p(y^*|\mathbf{x}^*, \mathcal{S}_{\text{train}}) = \int p(y^*|\mathbf{x}^*, \theta) p(\theta|\mathcal{S}_{\text{train}}) d\theta \tag{10.11}$$

Ultimately, the success of FSL hinges on leveraging shared structure across tasks, designing effective adaptation mechanisms, and optimizing inductive biases to mitigate data scarcity.

10.1 Meta-learning formulation in Few Shot Learning

Meta-learning, particularly in the context of Few-Shot Learning (FSL), aims to train a model such that it can generalize effectively to novel tasks with very few training examples. This is accomplished by optimizing the learning process itself, enabling the model to rapidly adapt to new tasks. Given a distribution over tasks $p(\mathcal{T})$, where each task $\mathcal{T}_i$ is characterized by a small support set $S_i = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^K$ and a corresponding query set Q_i, the objective of meta-learning is to train a model f_θ with parameters θ such that it minimizes the expected loss over new tasks drawn from $p(\mathcal{T})$.

$$\min_\theta \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}_\mathcal{T}(f_\theta) \right] \tag{10.12}$$

where $\mathcal{L}_\mathcal{T}(f_\theta)$ represents the loss function associated with the task $\mathcal{T}$, typically computed over the query set after adapting the model using the support set. One of the most widely used meta-learning approaches is Model-Agnostic Meta-Learning (MAML), which optimizes for an initial parameter set θ that can be quickly adapted to a new task using only a few gradient steps. The inner-loop adaptation updates the model parameters as:

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{10.13}$$

where α is the inner-loop learning rate. The meta-objective then minimizes the loss computed on the query set with respect to the updated parameters:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \tag{10.14}$$

which requires computing the meta-gradient:

$$\nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}) \tag{10.15}$$

leading to the final meta-update:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \tag{10.16}$$

where β is the meta-learning rate. Another formulation follows metric-based meta-learning, where the model learns an embedding function g_θ that maps input samples to a feature space where classification can be performed using a simple distance metric. In Prototypical Networks, the class prototype for each class c in the support set is computed as:

$$\mathbf{c}_c = \frac{1}{|S_c|} \sum_{(\mathbf{x}_j, y_j) \in S_c} g_\theta(\mathbf{x}_j) \tag{10.17}$$

where S_c is the set of examples belonging to class c. The probability of assigning a query sample $\mathbf{x}_q$ to class c is then given by:

$$p(y_q = c \mid \mathbf{x}_q) = \frac{\exp(-d(g_\theta(\mathbf{x}_q), \mathbf{c}_c))}{\sum_{c'} \exp(-d(g_\theta(\mathbf{x}_q), \mathbf{c}_{c'}))} \tag{10.18}$$

where $d(\cdot, \cdot)$ is a distance metric, often chosen as the squared Euclidean distance. Contrastive learning-based meta-learning methods instead optimize a loss function based on relative similarities. A common choice is the contrastive loss:

$$\mathcal{L} = \sum_{i,j} \mathbb{I}[y_i = y_j] \cdot d(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j)) - (1 - \mathbb{I}[y_i = y_j]) \cdot \max(0, m - d(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j))) \tag{10.19}$$

where m is a margin parameter and $\mathbb{I}[\cdot]$ is an indicator function. Bayesian meta-learning approaches introduce a probabilistic treatment where the model parameters θ are sampled from a learned posterior distribution $p(\theta \mid D)$, updated via variational inference:

$$q_\phi(\theta) = \arg\min_q D_{KL}(q(\theta) \parallel p(\theta \mid D)) \tag{10.20}$$

where $D_{KL}(\cdot \parallel \cdot)$ denotes the Kullback-Leibler divergence. The expected posterior predictive distribution for a new task is then computed as:

$$p(y_q \mid \mathbf{x}_q, S) = \mathbb{E}_{\theta \sim q_\phi(\theta)}[p(y_q \mid \mathbf{x}_q, \theta)] \tag{10.21}$$

which can be approximated using Monte Carlo sampling. Gradient-based meta-learning methods can also be formulated using recurrent neural networks, where the meta-learner is a recurrent model that updates a task-specific hidden state h_t over a sequence of examples. Given a sequence of support set updates, the parameterized recurrence is given by:

$$h_t = f_\psi(h_{t-1}, \nabla_\theta \mathcal{L}_t) \tag{10.22}$$

and the model parameters for a given task are inferred as:

$$\theta_t = g_\psi(h_t) \tag{10.23}$$

where f_ψ and g_ψ are recurrent function approximators parameterized by ψ. These meta-learning paradigms share a common objective: to minimize the expected generalization error of a model trained over a distribution of tasks, enabling rapid adaptation to new scenarios with minimal labeled data. The fundamental mechanism across formulations involves optimizing a bi-level objective where the outer loop adjusts meta-parameters and the inner loop performs task-specific adaptation. The optimization problem governing this framework can be expressed in a general form as:

$$\min_\theta \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}_\mathcal{T}(f_{\theta - \nabla_\theta \mathcal{L}_\mathcal{T}(f_\theta)}) \right] \tag{10.24}$$

which encapsulates the core philosophy of meta-learning: learning to learn.

10.2 Model-Agnostic Meta-Learning (MAML) in Few Shot Learning

Model-Agnostic Meta-Learning (MAML) is a fundamental optimization-based approach in few-shot learning, designed to enable models to quickly adapt to new tasks with minimal data. The core idea is to learn an initialization of model parameters that facilitates rapid adaptation through gradient-based updates. Given a distribution over tasks, denoted as $p(\mathcal{T})$, MAML optimizes for a set of parameters θ such that for a new task $\mathcal{T}_i \sim p(\mathcal{T})$, the adapted parameters θ_i' after a small number of gradient steps yield high performance on the task.

For a given task $\mathcal{T}_i$, we assume a dataset $\mathcal{D}_i$ composed of a support set $\mathcal{D}_i^{\text{train}}$ and a query set $\mathcal{D}_i^{\text{test}}$. Let the model be parameterized by θ and let the task-specific loss function be $\mathcal{L}_{\mathcal{T}_i}$. The adaptation to task $\mathcal{T}_i$ involves performing one or more gradient descent steps with respect to $\mathcal{L}_{\mathcal{T}_i}$ evaluated on $\mathcal{D}_i^{\text{train}}$. The standard first-order adaptation rule for a single gradient step is given by:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}) \tag{10.25}$$

where α is the learning rate for task-specific updates. The meta-objective is to find an initialization θ such that after task-specific adaptation, the model performs well on the query set $\mathcal{D}_i^{\text{test}}$. The meta-loss is computed over the query set:

$$\mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta_i', \mathcal{D}_i^{\text{test}}) \tag{10.26}$$

which leads to the outer-level optimization of θ:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta_i', \mathcal{D}_i^{\text{test}}) \tag{10.27}$$

where β is the meta-learning rate. Expanding the gradient term using the chain rule,

$$\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta_i', \mathcal{D}_i^{\text{test}}) = \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_i}(\theta_i', \mathcal{D}_i^{\text{test}}) \cdot \frac{d\theta_i'}{d\theta} \tag{10.28}$$

with

$$\frac{d\theta_i'}{d\theta} = I - \alpha \nabla_\theta^2 \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}) \tag{10.29}$$

where I is the identity matrix and $\nabla_\theta^2 \mathcal{L}_{\mathcal{T}_i}$ is the Hessian of the loss function with respect to θ. The optimization process involves computing second-order derivatives, which can be computationally expensive. A first-order approximation, called First-Order MAML (FOMAML), simplifies this update by ignoring the Hessian term:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_i}(\theta_i', \mathcal{D}_i^{\text{test}}) \tag{10.30}$$

where θ_i' is still obtained using the standard gradient descent update. This approximation reduces computational overhead while still achieving strong meta-learning performance. The effectiveness of MAML depends on its ability to learn an initialization θ that allows rapid adaptation across a diverse set of tasks. Formally, we define the expected meta-loss over the task distribution:

$$\mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}_i}(\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}), \mathcal{D}_i^{\text{test}}) \right] \tag{10.31}$$

which represents the expectation over the loss incurred after a single adaptation step. The meta-optimization aims to minimize this expected loss, ensuring that θ provides an effective starting point for task-specific fine-tuning. For multiple gradient adaptation steps, the parameter update for a given task follows an iterative process:

$$\theta_i^{(k)} = \theta_i^{(k-1)} - \alpha \nabla_{\theta_i^{(k-1)}} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(k-1)}, \mathcal{D}_i^{\text{train}}) \tag{10.32}$$

for $k = 1, \ldots, K$, where K denotes the number of inner-loop gradient updates. The final adapted parameters $\theta_i^{(K)}$ are then used to compute the meta-loss:

$$\mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(K)}, \mathcal{D}_i^{\text{test}}) \tag{10.33}$$

which determines the outer-loop optimization of θ. The full meta-gradient computation involves backpropagating through K gradient updates, leading to higher-order derivative terms:

$$\nabla_\theta \mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta_i^{(K)}} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(K)}, \mathcal{D}_i^{\text{test}}) \cdot \prod_{k=1}^{K} \frac{d\theta_i^{(k)}}{d\theta_i^{(k-1)}} \tag{10.34}$$

where each term captures the second-order effects of iterative gradient updates.

10.3 Metric-based learning in Few Shot Learning

Metric-based learning in the context of few-shot learning is a paradigm that relies on learning a similarity metric to compare query examples against a set of labeled support examples. Given a support set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$, where each x_i is an input instance and y_i is its corresponding label, and a query instance x_q, the goal is to predict y_q based on a learned metric function $d_\theta(x_q, x_i)$ parameterized by θ. The fundamental principle is that instances belonging to the same class should be mapped closer together in an embedding space, while instances from different classes should be mapped further apart. Mathematically, this can be formulated as minimizing the intra-class variance while maximizing the inter-class variance, which can be written as

$$\min_\theta \sum_{(x_i,y_i)\in\mathcal{S}} \sum_{(x_j,y_j)\in\mathcal{S}} \mathbb{I}(y_i = y_j) d_\theta(x_i, x_j) - \lambda \sum_{(x_i,y_i)\in\mathcal{S}} \sum_{(x_j,y_j)\in\mathcal{S}} \mathbb{I}(y_i \neq y_j) d_\theta(x_i, x_j), \tag{10.35}$$

where λ is a weighting factor that controls the relative importance of inter-class separation. The metric function $d_\theta(x_i, x_j)$ is often implemented using a neural network that maps instances into an embedding space via an encoder $f_\theta : \mathcal{X} \to \mathbb{R}^d$, such that

$$d_\theta(x_i, x_j) = \|f_\theta(x_i) - f_\theta(x_j)\|^2. \tag{10.36}$$

One of the key approaches in metric-based few-shot learning is prototypical networks, where each class in the support set is represented by a prototype c_k, computed as the mean of embedded support examples belonging to class k:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i,y_i)\in S_k} f_\theta(x_i). \tag{10.37}$$

The classification of a query example x_q is then performed by computing its similarity to each prototype, typically using the squared Euclidean distance,

$$p(y_q = k|x_q) = \frac{\exp(-d(f_\theta(x_q), c_k))}{\sum_{k'} \exp(-d(f_\theta(x_q), c_{k'}))}. \tag{10.38}$$

Another prominent approach is relation networks, which replace the explicit distance metric with a learned similarity function $g_\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, parameterized by ϕ. The probability of assigning query x_q to class k is given by

$$p(y_q = k|x_q) = \frac{\exp(g_\phi(f_\theta(x_q), c_k))}{\sum_{k'} \exp(g_\phi(f_\theta(x_q), c_{k'}))}. \tag{10.39}$$

Siamese networks approach metric learning through pairwise comparisons, where a shared encoder f_θ maps pairs of examples (x_i, x_j) into an embedding space, and the similarity is computed as

$$s_\theta(x_i, x_j) = \sigma(W \cdot |f_\theta(x_i) - f_\theta(x_j)|), \tag{10.40}$$

where W is a learned weight vector and σ is the sigmoid activation function. The model is trained to minimize the binary cross-entropy loss,

$$\mathcal{L} = - \sum_{(x_i,y_i),(x_j,y_j)\in\mathcal{S}} \left[\mathbb{I}(y_i = y_j) \log s_\theta(x_i, x_j) + (1 - \mathbb{I}(y_i = y_j)) \log(1 - s_\theta(x_i, x_j))\right]. \tag{10.41}$$

Triplet networks extend this by considering an anchor example x_a, a positive example x_p (same class), and a negative example x_n (different class), enforcing the constraint

$$d(f_\theta(x_a), f_\theta(x_p)) + m < d(f_\theta(x_a), f_\theta(x_n)), \tag{10.42}$$

where m is a margin hyperparameter. The loss function is given by

$$\mathcal{L} = \sum_{\text{triplets}} \max\big(0, d(f_\theta(x_a), f_\theta(x_p)) - d(f_\theta(x_a), f_\theta(x_n)) + m\big). \tag{10.43}$$

By optimizing these loss functions, metric-based few-shot learning methods enable models to generalize well to novel classes, as the learned metric encodes class-invariant representations of instances. The embedding space acts as a structured manifold where geometric distances capture semantic similarities, allowing rapid adaptation to unseen tasks with minimal labeled data.

10.4 Bayesian methods in Few Shot Learning

Bayesian methods in Few-Shot Learning (FSL) provide a probabilistic framework to model uncertainty in learning from a small number of examples. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with a limited number of training examples N, Bayesian approaches aim to infer a distribution over the model parameters θ, rather than a single deterministic estimate, allowing the model to generalize effectively despite limited data. The core idea is to use Bayes' theorem to compute the posterior distribution over θ:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{10.44}$$

where $p(\mathcal{D}|\theta)$ is the likelihood of the data given parameters θ, $p(\theta)$ is the prior over the parameters, and $p(\mathcal{D})$ is the marginal likelihood or evidence. The posterior distribution encapsulates the uncertainty in parameter estimation, making it particularly useful for FSL where the data is scarce.

The Bayesian framework enables meta-learning by assuming that tasks in FSL are drawn from an unknown distribution over tasks $p(\mathcal{T})$. Given a set of tasks $\{\mathcal{T}_i\}$, each with a small support set $\mathcal{S}_i$ and query set $\mathcal{Q}_i$, the goal is to learn a distribution over task-specific parameters θ_i conditioned on the support set:

$$p(\theta_i|\mathcal{S}_i) = \frac{p(\mathcal{S}_i|\theta_i)p(\theta_i)}{p(\mathcal{S}_i)} \tag{10.45}$$

A hierarchical Bayesian model treats the task-specific parameters θ_i as drawn from a global distribution with hyperparameters ϕ:

$$p(\theta_i|\phi) = \mathcal{N}(\mu_\phi, \Sigma_\phi) \tag{10.46}$$

where $\phi = (\mu_\phi, \Sigma_\phi)$ defines a prior distribution over θ_i. This enables the model to generalize across tasks by updating the prior $p(\phi)$ using multiple tasks:

$$p(\phi|\{\mathcal{S}_i\}) \propto p(\{\mathcal{S}_i\}|\phi)p(\phi) \tag{10.47}$$

Given a new task with support set $\mathcal{S}_*$, the predictive distribution over labels y_* for query inputs x_* is obtained by marginalizing over θ:

$$p(y_*|x_*, \mathcal{S}_*) = \int p(y_*|x_*, \theta)p(\theta|\mathcal{S}_*)d\theta \tag{10.48}$$

Since exact inference of $p(\theta|\mathcal{D})$ is often intractable, variational inference is commonly used. We approximate $p(\theta|\mathcal{D})$ with a variational distribution $q(\theta|\lambda)$, parameterized by λ, minimizing the Kullback-Leibler (KL) divergence:

$$\mathcal{L}(\lambda) = \mathbb{E}_{q(\theta|\lambda)}[\log p(\mathcal{D}|\theta)] - D_{KL}(q(\theta|\lambda)\|p(\theta)) \tag{10.49}$$

This leads to an optimization problem where λ is updated via gradient descent to minimize $\mathcal{L}(\lambda)$, ensuring the learned distribution $q(\theta|\lambda)$ approximates the true posterior. Another approach is Bayesian nonparametrics, where the distribution over functions $f : \mathcal{X} \to \mathcal{Y}$ is modeled using a Gaussian Process (GP), providing a principled way to quantify uncertainty. Given a prior GP:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{10.50}$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function, the posterior over functions given training data $\mathcal{D}$ is:

$$p(f|\mathcal{D}) = \mathcal{GP}(\tilde{m}(x), \tilde{k}(x, x')) \tag{10.51}$$

where $\tilde{m}(x)$ and $\tilde{k}(x, x')$ are the posterior mean and covariance functions derived using Bayes' rule. The predictive distribution for a new input x_* follows:

$$p(y_*|x_*, \mathcal{D}) = \int p(y_*|f(x_*))p(f|\mathcal{D})df \tag{10.52}$$

which yields a closed-form Gaussian predictive distribution due to the properties of GPs. Another key Bayesian technique is Bayesian Neural Networks (BNNs), where instead of learning a single weight matrix W, a distribution over weights is maintained:

$$p(W|\mathcal{D}) \propto p(\mathcal{D}|W)p(W) \tag{10.53}$$

Inference in BNNs requires approximations like Monte Carlo Dropout, where approximate Bayesian inference is performed by applying dropout at both training and test time, yielding an empirical posterior:

$$p(y_*|x_*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y_*|x_*, W_t) \tag{10.54}$$

where W_t are sampled from the variational posterior $q(W)$. This captures uncertainty in predictions, crucial for robust few-shot learning. Bayesian optimization further refines FSL by selecting informative examples to maximize learning efficiency. Given an acquisition function $a(x)$ based on the posterior predictive distribution:

$$x^* = \arg\max_x a(x) \tag{10.55}$$

a new query point is selected to minimize uncertainty and maximize information gain, enhancing generalization from few samples. Thus, Bayesian methods in FSL provide a mathematically rigorous probabilistic framework to model uncertainty, transfer knowledge across tasks, and make robust predictions even with limited training data.

11 Metric Learning

Metric learning is a fundamental concept in machine learning and mathematical optimization, concerned with learning a distance function or similarity measure that captures the underlying structure of data. Given a set of training samples $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$, where each $\mathbf{x}_i \in \mathbb{R}^d$, the objective is to learn a metric function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ that quantifies the dissimilarity between any two points. A standard choice for $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Mahalanobis distance, which generalizes the Euclidean distance and is defined as

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)} \tag{11.1}$$

where $\mathbf{M}$ is a positive semi-definite matrix, i.e., $\mathbf{M} \succeq 0$, ensuring that the distance satisfies the properties of a metric. Learning the matrix $\mathbf{M}$ is the core challenge in metric learning. When $\mathbf{M} = \mathbf{I}$, the metric reduces to the standard Euclidean distance:

$$d_{\mathbf{I}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} \tag{11.2}$$

However, metric learning seeks to optimize $\mathbf{M}$ such that semantically similar data points have smaller distances than dissimilar points. This is typically formulated as an optimization problem where we enforce constraints on distances between pairs of points. Given a set of pairs $\mathcal{S}$ (similar pairs) and $\mathcal{D}$ (dissimilar pairs), the goal is to satisfy

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq u, \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S} \tag{11.3}$$

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \geq l, \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D} \tag{11.4}$$

for some thresholds u and l. One widely used approach to metric learning is Large Margin Nearest Neighbors (LMNN), which seeks to optimize $\mathbf{M}$ by minimizing a hinge loss on triplets of points $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$, where $\mathbf{x}_i$ is closer to $\mathbf{x}_j$ than $\mathbf{x}_k$, i.e.,

$$\sum_{(i,j,k)} \max(0, 1 + d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k)) \tag{11.5}$$

where the triplet loss ensures that the distance between similar points is smaller than the distance between dissimilar ones by a margin of at least 1. An alternative approach is the information-theoretic metric learning (ITML) framework, which minimizes the Kullback-Leibler divergence between two Gaussian distributions defined by different Mahalanobis distances:

$$D_{\mathrm{KL}}(p_{\mathbf{M}} \| p_{\mathbf{M}_0}) = \frac{1}{2} \left(\mathrm{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) - d \right) \tag{11.6}$$

Deep metric learning extends these concepts by parameterizing the metric function using a neural network f_θ with learnable parameters θ, leading to a learned embedding space where distances are computed as

$$d_\theta(\mathbf{x}_i, \mathbf{x}_j) = \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|^2 \tag{11.7}$$

Instead of learning a linear transformation as in Mahalanobis-based methods, deep metric learning implicitly learns a nonlinear mapping such that the Euclidean distance in the transformed space aligns with semantic similarity. The contrastive loss is commonly used in this context:

$$\sum_{(i,j)\in\mathcal{S}} d_\theta(\mathbf{x}_i, \mathbf{x}_j) + \sum_{(i,j)\in\mathcal{D}} \max(0, m - d_\theta(\mathbf{x}_i, \mathbf{x}_j))^2 \tag{11.8}$$

Alternatively, the triplet loss is defined as

$$\sum_{(i,j,k)} \max(0, d_\theta(\mathbf{x}_i, \mathbf{x}_j) - d_\theta(\mathbf{x}_i, \mathbf{x}_k) + m) \tag{11.9}$$

A more recent approach, the normalized temperature-scaled cross-entropy loss (NT-Xent), utilizes a softmax formulation:

$$\ell = -\sum_i \log \frac{\exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k\neq i} \exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{11.10}$$

where $\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j)$ is the cosine similarity,

$$\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j) = \frac{\mathbf{z}_i^T \mathbf{z}_j}{\|\mathbf{z}_i\| \|\mathbf{z}_j\|} \tag{11.11}$$

By optimizing the metric under these constraints, metric learning produces feature representations that are well-structured for downstream tasks such as retrieval, verification, and clustering.

11.1 Large Margin Nearest Neighbors (LMNN) approach to Metric Learning

The **Large Margin Nearest Neighbors (LMNN)** approach to metric learning is a method that aims to learn a **Mahalanobis distance metric** that optimizes nearest neighbor classification by ensuring that points belonging to the same class remain close while different-class points are separated by a large margin. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents a feature vector and y_i denotes its class label, LMNN learns a transformation matrix $\mathbf{L} \in \mathbb{R}^{d\times d}$ such that the squared distance between any two points is given by

$$d_{\mathbf{L}}^2(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) \tag{11.12}$$

where $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ is a positive semi-definite (PSD) matrix. The optimization problem is designed to minimize classification error by enforcing that each sample $\mathbf{x}_i$ remains close to its **target neighbors**, denoted by $\mathcal{N}_i$, while keeping different-class points at a distance. The objective function consists of two competing terms: a **pull term** that encourages proximity among target neighbors and a **push term** that enforces a large margin separation from impostors (samples of different classes that intrude upon the local neighborhood). The **pull term** is formulated as

$$\sum_i \sum_{j\in\mathcal{N}_i} d_{\mathbf{L}}^2(\mathbf{x}_i, \mathbf{x}_j) = \sum_i \sum_{j\in\mathcal{N}_i} (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j). \tag{11.13}$$

This term ensures that the squared distances between each point and its target neighbors remain small. The **push term**, on the other hand, ensures that impostors—points $\mathbf{x}_k$ of a different class than $\mathbf{x}_i$ but closer than its farthest target neighbor—are pushed away by at least a margin 1, formulated using hinge loss as

$$\sum_i \sum_{j\in\mathcal{N}_i} \sum_k \xi_{ijk} \tag{11.14}$$

subject to the constraints

$$d_{\mathbf{L}}^2(\mathbf{x}_i, \mathbf{x}_k) \geq d_{\mathbf{L}}^2(\mathbf{x}_i, \mathbf{x}_j) + 1 - \xi_{ijk}, \quad \xi_{ijk} \geq 0. \tag{11.15}$$

The total LMNN objective function combines these terms as

$$\min_{\mathbf{M}\succeq 0} \sum_i \sum_{j\in\mathcal{N}_i} d_{\mathbf{L}}^2(\mathbf{x}_i, \mathbf{x}_j) + C \sum_i \sum_{j\in\mathcal{N}_i} \sum_k \xi_{ijk} \tag{11.16}$$

where C is a regularization parameter controlling the tradeoff between the pull and push terms. The constraint $\mathbf{M} \succeq 0$ ensures that the learned metric remains a valid Mahalanobis distance. The optimization is typically solved using semidefinite programming (SDP) or projected gradient descent methods while enforcing the PSD constraint on $\mathbf{M}$. To interpret the impact of the learned metric, consider an eigenvalue decomposition of $\mathbf{M}$, given by

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \tag{11.17}$$

where $\mathbf{U}$ is an orthonormal matrix and $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues. The transformation $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}^{1/2}$ effectively maps the original data space into a new space where distances are reshaped to optimize nearest-neighbor classification performance. The LMNN framework ensures that different classes are well-separated while preserving local neighborhood structures, leading to improved generalization in classification tasks.

11.2 Information-theoretic metric learning (ITML) framework approach to Metric Learning

The Information-Theoretic Metric Learning (ITML) framework is a fundamental approach to metric learning that formulates the problem as an optimization task driven by information theory principles, particularly Kullback-Leibler (KL) divergence minimization. Given a set of data points $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$, the goal of ITML is to learn a Mahalanobis distance metric parameterized by a positive semi-definite (PSD) matrix $\mathbf{M}$, so that distances between similar points are minimized while distances between dissimilar points are maintained above a certain threshold. The Mahalanobis distance between two points x_i and x_j under the learned metric $\mathbf{M}$ is defined as

$$d_{\mathbf{M}}(x_i, x_j) = \sqrt{(x_i - x_j)^T \mathbf{M} (x_i - x_j)} \tag{11.18}$$

where $\mathbf{M}$ is a symmetric positive semi-definite matrix, i.e., $\mathbf{M} \succeq 0$, ensuring that $d_{\mathbf{M}}(x_i, x_j)$ defines a proper distance function. The fundamental idea of ITML is to find a metric $\mathbf{M}$ that minimizes the divergence from a given prior metric $\mathbf{M}_0$, typically the identity matrix $\mathbf{I}$, subject to constraints that enforce distances between pairs of points to satisfy specified conditions. To formalize this, ITML minimizes the KL divergence between the distributions parameterized by $\mathbf{M}$ and $\mathbf{M}_0$. Given that Gaussian distributions parameterized by a Mahalanobis metric have a natural representation in terms of covariance matrices, the KL divergence between two Gaussian distributions with covariance matrices $\mathbf{M}^{-1}$ and $\mathbf{M}_0^{-1}$ is

$$D_{\mathrm{KL}}(\mathcal{N}(0, \mathbf{M}^{-1}) \| \mathcal{N}(0, \mathbf{M}_0^{-1})) = \frac{1}{2}\left(\operatorname{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) - d\right) \tag{11.19}$$

where $\operatorname{tr}(\cdot)$ denotes the trace operator, and $\log\det(\cdot)$ is the logarithm of the determinant. The optimization problem in ITML is thus formulated as

$$\min_{\mathbf{M}\succeq 0} \quad \operatorname{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) \tag{11.20}$$

subject to pairwise constraints of the form

$$(x_i - x_j)^T \mathbf{M} (x_i - x_j) \le u, \quad \forall (x_i, x_j) \in \mathcal{S} \tag{11.21}$$

$$(x_i - x_j)^T \mathbf{M} (x_i - x_j) \ge \ell, \quad \forall (x_i, x_j) \in \mathcal{D} \tag{11.22}$$

where $\mathcal{S}$ and $\mathcal{D}$ denote sets of similar and dissimilar pairs, respectively, and u, ℓ are predefined thresholds that control the tightness of the constraints. The use of KL divergence as the objective function ensures that the learned metric remains close to the prior $\mathbf{M}_0$ while satisfying the constraints. To solve this optimization problem, a log-barrier method is commonly employed, leading to an iterative update for $\mathbf{M}$ of the form

$$\mathbf{M}^{(t+1)} = \mathbf{M}^{(t)} + \gamma_t \mathbf{M}^{(t)}(x_i - x_j)(x_i - x_j)^T \mathbf{M}^{(t)} \tag{11.23}$$

where γ_t is an adaptive step size chosen to satisfy the constraints. This iterative procedure ensures that $\mathbf{M}$ remains positive semi-definite at every step. The learned metric is influenced by the data-driven constraints while maintaining a balance between adaptation and preservation of the prior structure. The ITML framework exhibits strong theoretical properties, particularly in terms of generalization and convexity. Since the optimization problem is convex in $\mathbf{M}$, the solution is globally optimal, ensuring robustness. Moreover, the constraint formulation allows for a natural interpretation: if the prior $\mathbf{M}_0 = \mathbf{I}$, then the learned metric is a transformation of Euclidean space that deforms distances according to the prescribed similarity relationships. In practice, ITML is highly effective due to its ability to incorporate prior knowledge and adapt flexibly to data distributions. The reliance on KL divergence ensures stability in learning, preventing excessive deviation from the prior. By iteratively refining the metric with convex optimization, ITML efficiently learns a Mahalanobis metric that best captures the underlying relationships within the data.

11.3 Deep Metric Learning

Deep Metric Learning (DML) is a fundamental approach in machine learning that aims to learn an embedding function $f_\theta : \mathbb{R}^d \to \mathbb{R}^m$ that maps input data points from a high-dimensional space $\mathbb{R}^d$ to a lower-dimensional space $\mathbb{R}^m$, such that semantically similar points are brought closer together while dissimilar points are pushed farther apart. Formally, given a dataset $\mathcal{X} = \{x_i\}_{i=1}^N$ with labels $\mathcal{Y} = \{y_i\}_{i=1}^N$, the goal is to optimize the parameters θ of the embedding function f_θ such that for any triplet of points (x_a, x_p, x_n) where x_a (anchor) and x_p (positive) belong to the same class $(y_a = y_p)$, while x_n (negative) belongs to a different class $(y_a \neq y_n)$, the Euclidean distance in the embedding space satisfies

$$\|f_\theta(x_a) - f_\theta(x_p)\|_2^2 + \alpha < \|f_\theta(x_a) - f_\theta(x_n)\|_2^2, \tag{11.24}$$

where $\alpha > 0$ is a margin that enforces a minimum separation between positive and negative pairs. To achieve this objective, DML often employs contrastive loss, which minimizes the Euclidean distance between positive pairs while maximizing it for negative pairs. Given a pair (x_i, x_j), the contrastive loss function is

$$\mathcal{L}_{\text{contrastive}} = (1 - y_{ij}) \max(0, m - d_{ij})^2 + y_{ij} d_{ij}^2, \tag{11.25}$$

where $y_{ij} = 1$ if x_i and x_j belong to the same class, $y_{ij} = 0$ otherwise, $d_{ij} = \|f_\theta(x_i) - f_\theta(x_j)\|_2$ is the Euclidean distance, and m is a margin parameter. Another widely used loss function in DML is the triplet loss, given by

$$\mathcal{L}_{\text{triplet}} = \sum_{(a,p,n)\in\mathcal{T}} \max(0, \|f_\theta(x_a) - f_\theta(x_p)\|_2^2 - \|f_\theta(x_a) - f_\theta(x_n)\|_2^2 + \alpha). \tag{11.26}$$

This loss function ensures that the anchor-positive distance is always smaller than the anchor-negative distance by at least α, thereby enforcing clustering of similar instances while separating dissimilar ones. In deep learning-based metric learning, the embedding function f_θ is typically parameterized by a deep neural network such as a convolutional neural network (CNN) for image data or a recurrent neural network (RNN) for sequential data. Let x be an input sample and let the network be denoted as $f_\theta(x)$. The network is optimized by minimizing the empirical risk

$$\mathcal{L}(\theta) = \mathbb{E}_{(x_a,x_p,x_n)\sim\mathcal{D}} \mathcal{L}_{\text{triplet}}(x_a, x_p, x_n), \tag{11.27}$$

where $\mathcal{D}$ represents the data distribution. To further refine the embeddings, methods such as hard negative mining are employed, where the negative samples are chosen such that

$$d_{an} = \min_{n \in \mathcal{N}_a} \|f_\theta(x_a) - f_\theta(x_n)\|_2. \tag{11.28}$$

This strategy ensures that the network focuses on difficult examples that are closer to the anchor, making learning more effective. To generalize beyond triplet-based losses, one can also consider the N-pair loss, which extends the triplet loss by incorporating multiple negatives:

$$\mathcal{L}_{\text{N-pair}} = \sum_{i=1}^{N} \log \left(1 + \sum_{j \neq i} \exp(\|f_\theta(x_i) - f_\theta(x_j)\|_2^2 - \|f_\theta(x_i) - f_\theta(x_i^+)\|_2^2) \right), \tag{11.29}$$

where x_i^+ is a positive sample corresponding to x_i. An alternative formulation is Proxy-based Deep Metric Learning, where trainable class proxies p_c are introduced for each class c, and each input is encouraged to be close to its respective proxy while remaining far from proxies of other classes. The softmax-based proxy loss is

$$\mathcal{L}_{\text{proxy}} = -\sum_{i=1}^{N} \log \frac{\exp(-\|f_\theta(x_i) - p_{y_i}\|_2^2)}{\sum_c \exp(-\|f_\theta(x_i) - p_c\|_2^2)}. \tag{11.30}$$

Another recent improvement in deep metric learning involves contrastive learning-based methods, where positive and negative pairs are dynamically generated using a memory bank or momentum encoder. The InfoNCE loss, a contrastive loss derived from mutual information maximization, is

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{\exp(f_\theta(x_i) \cdot f_\theta(x_i^+)/\tau)}{\sum_j \exp(f_\theta(x_i) \cdot f_\theta(x_j^-)/\tau)} \right], \tag{11.31}$$

where τ is a temperature parameter that controls the distribution of distances. Furthermore, a probabilistic perspective can be adopted, where embeddings are modeled as distributions rather than point estimates. This leads to the deep variational metric learning approach, where embeddings are represented as Gaussian distributions $q_\theta(z|x)$, and the distance metric is defined using the Kullback-Leibler divergence

$$D_{\text{KL}}(q_\theta(z|x_i) \| q_\theta(z|x_j)). \tag{11.32}$$

The corresponding loss function is formulated as

$$\mathcal{L}_{\text{variational}} = \sum_{i,j} y_{ij} D_{\text{KL}}(q_\theta(z|x_i) \| q_\theta(z|x_j)) + (1 - y_{ij}) \max(0, m - D_{\text{KL}}(q_\theta(z|x_i) \| q_\theta(z|x_j))). \tag{11.33}$$

Finally, the quality of the learned embeddings can be evaluated using retrieval-based metrics such as mean average precision (mAP), recall at k (R@k), and normalized mutual information (NMI), ensuring that the learned distance metric effectively captures semantic similarity.

11.4 Normalized temperature-scaled cross-entropy loss (NT-Xent) approach to Metric Learning

The Normalized Temperature-scaled Cross-Entropy Loss (NT-Xent) is a fundamental loss function in contrastive learning, particularly in self-supervised representation learning. The NT-Xent loss is designed to maximize agreement between similar (positive) pairs while pushing dissimilar (negative) pairs apart in the learned feature space. Mathematically, it builds upon softmax-based similarity scoring and temperature scaling, ensuring a well-calibrated probabilistic interpretation of similarity.

Given a batch of N samples, let each data point x_i be augmented to obtain two correlated views, denoted as x'_i and x''_i. These views are encoded into the feature space using a learned function $f_\theta(\cdot)$, which is typically a deep neural network. The feature representations of the two augmented views are given by

$$z'_i = f_\theta(x'_i), \quad z''_i = f_\theta(x''_i) \tag{11.34}$$

where the output embeddings are L2-normalized to lie on a unit hypersphere:

$$\|z'_i\| = 1, \quad \|z''_i\| = 1. \tag{11.35}$$

The similarity between two representations is computed using the cosine similarity, which is given by

$$\text{sim}(z_i, z_j) = \frac{z_i \cdot z_j}{\|z_i\|\|z_j\|} = z_i^\top z_j. \tag{11.36}$$

Since z_i and z_j are unit vectors, their dot product is directly the cosine of the angle between them. To define the NT-Xent loss, a temperature scaling parameter τ is introduced to control the sharpness of the similarity distribution. The temperature-scaled similarity is given by

$$s_{ij} = \frac{z_i^\top z_j}{\tau}. \tag{11.37}$$

This scaling prevents collapse by adjusting the entropy of the softmax distribution. The probability of z'_i matching with z''_i (its positive counterpart) over all possible samples in the batch is computed using the softmax function

$$p_{i,i''} = \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})}. \tag{11.38}$$

The NT-Xent loss for a single positive pair (z'_i, z''_i) is then formulated as

$$\ell_i = -\log p_{i,i''} = -\log \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})}. \tag{11.39}$$

To compute the full batch-wise NT-Xent loss, we sum over all samples and their corresponding augmentations, leading to

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} (\ell_i + \ell_{i''}) = -\frac{1}{2N} \sum_{i=1}^{N} \left(\log \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})} + \log \frac{\exp(s_{i'',i})}{\sum_{j \neq i} \exp(s_{i'',j})} \right). \tag{11.40}$$

The denominator in the softmax function acts as a contrastive term, where all other embeddings in the batch contribute as negative samples. Since the numerator contains only one positive pair, the loss forces these representations to be closer, while the denominator encourages separation from all other samples. A key insight into NT-Xent is its connection to InfoNCE loss (used in contrastive predictive coding), but it is fully symmetric, meaning both views of the same instance contribute equally. This symmetry helps in learning invariant representations by reinforcing consistent feature extraction across different augmentations. To ensure proper gradient flow, the temperature τ is crucial. If τ is too high, the softmax distribution becomes too uniform, leading to weak discrimination. Conversely, if τ is too low, the model learns overly sharp distributions, which can lead to overfitting. The gradient of NT-Xent loss with respect to the embedding z_i can be derived using the softmax gradient formula

$$\frac{\partial \mathcal{L}}{\partial z_i} = \sum_j p_{i,j}(z_j - z_i). \tag{11.41}$$

This update step ensures that positive pairs $(z_i, z_{i''})$ get pulled together while negative pairs are pushed apart in proportion to their softmax probabilities. In the large batch limit, NT-Xent can be approximated by **Monte Carlo sampling** of negative pairs, ensuring computational efficiency in practical self-supervised learning frameworks such as **SimCLR**. Thus, the NT-Xent loss serves as a fundamental tool for learning **discriminative, invariant, and well-clustered embeddings** in self-supervised contrastive learning frameworks.

12 Adversial Learning

Adversarial learning is a fundamental paradigm in machine learning where two models are trained in opposition to each other, leading to the generation of robust representations and improvements in generalization. The central concept underlying adversarial learning can be mathematically formulated in terms of a minimax optimization problem, where one model seeks to maximize an objective function while the other aims to minimize it. Formally, given a function $f(\theta)$ parameterized by θ, adversarial learning can be framed as

$$\min_{\theta} \max_{\delta} \mathcal{L}(f(\theta, x+\delta), y) \tag{12.1}$$

where $\mathcal{L}$ is the loss function, x is the input, y is the ground truth label, and δ represents an adversarial perturbation optimized to maximize the loss. The adversarial perturbation δ is constrained within a perturbation bound ϵ, ensuring that the perturbed example $x+\delta$ remains within a small neighborhood of x according to a given norm constraint:

$$\|\delta\|_p \leq \epsilon \tag{12.2}$$

where $\|\cdot\|_p$ represents the p-norm. The most common choice is the ℓ_∞-norm, which results in perturbations constrained as

$$\|\delta\|_\infty \leq \epsilon. \tag{12.3}$$

A fundamental technique in adversarial learning is the generation of adversarial examples using gradient-based methods. The Fast Gradient Sign Method (FGSM) computes adversarial perturbations using the sign of the gradient of the loss function with respect to the input:

$$\delta = \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x), y)). \tag{12.4}$$

This results in the adversarial example:

$$x' = x + \delta = x + \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x), y)). \tag{12.5}$$

A more refined approach is the Projected Gradient Descent (PGD) method, which iteratively updates the adversarial example using gradient ascent:

$$x_{t+1} = \Pi_{B_\epsilon(x)} \left(x_t + \alpha \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x_t), y)) \right), \tag{12.6}$$

where $\Pi_{B_\epsilon(x)}$ denotes projection onto the ℓ_∞-ball of radius ϵ around x, and α is the step size. The adversarial training process modifies the learning objective by incorporating adversarially generated examples into the training procedure, leading to the following robust optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f(\theta, x+\delta), y) \right], \tag{12.7}$$

where $\mathcal{D}$ represents the data distribution. This formulation ensures that the model is trained on worst-case perturbations, improving robustness against adversarial attacks. A generative approach to adversarial learning is encapsulated in Generative Adversarial Networks (GANs), where a generator G and a discriminator D are trained in a two-player minimax game:

$$\min_{G} \max_{D} \mathbb{E}_{x\sim p_{\text{data}}(x)} \left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)} \left[\log(1 - D(G(z)))\right]. \tag{12.8}$$

Here, the generator G aims to produce samples $G(z)$ that are indistinguishable from real samples, while the discriminator D seeks to correctly classify real and generated samples. The optimal discriminator is given by

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}. \tag{12.9}$$

The equilibrium of the game is attained when $p_G(x) = p_{\text{data}}(x)$, leading to the global minimum of the Jensen-Shannon divergence between real and generated distributions. Variants of adversarial learning include Wasserstein GANs (WGANs), which replace the standard GAN loss with the Wasserstein distance:

$$\min_G \max_{\|D\|_L \leq 1} \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))], \tag{12.10}$$

where D is constrained to be a 1-Lipschitz function. The Wasserstein distance provides a more stable training procedure by mitigating mode collapse. In reinforcement learning, adversarial learning is utilized in adversarial policy training, where an agent maximizes a reward function while an adversary perturbs the environment:

$$\max_{\pi} \min_{\mathcal{A}} \mathbb{E}_{\tau \sim P(\tau|\pi,\mathcal{A})} \left[R(\tau) \right]. \tag{12.11}$$

Here, π is the policy, $\mathcal{A}$ represents adversarial perturbations, and $R(\tau)$ is the reward function. Mathematically rigorous guarantees for adversarial robustness involve formulating certified defenses, which provide provable bounds on a model's robustness. For example, randomized smoothing transforms a classifier $f(x)$ into a smoothed classifier:

$$g(x) = \mathbb{E}_{\eta \sim \mathcal{N}(0,\sigma^2 I)}[f(x + \eta)]. \tag{12.12}$$

A theoretical guarantee on robustness is given by

$$\mathbb{P}[g(x) = g(x + \delta)] \geq 1 - 2\exp\left(-\frac{\|\delta\|_2^2}{2\sigma^2}\right). \tag{12.13}$$

These mathematical formulations illustrate the depth of adversarial learning, emphasizing its fundamental role in improving model robustness and stability.

12.1 Fast Gradient Sign Method (FGSM) in Adversial Learning

The Fast Gradient Sign Method (FGSM) is a crucial technique in adversarial learning, a domain that investigates the vulnerability of deep learning models to adversarial perturbations. Formally, given a neural network classifier $f(\mathbf{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, where $\mathbf{x} \in \mathbb{R}^d$ is an input and y is the corresponding true label, the objective of FGSM is to generate an adversarial example $\mathbf{x}_{\text{adv}}$ by perturbing $\mathbf{x}$ along the gradient direction that maximizes the classification loss. The adversarial perturbation is computed using the gradient of the loss function $J(\boldsymbol{\theta}, \mathbf{x}, y)$ with respect to the input $\mathbf{x}$, given by

$$\mathbf{g} = \nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y). \tag{12.14}$$

FGSM constructs an adversarial input by taking the sign of this gradient and scaling it by a perturbation magnitude ϵ, leading to the perturbed input

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \, \text{sign}(\mathbf{g}). \tag{12.15}$$

Mathematically, this attack is formulated as an optimization problem where the goal is to maximize the loss function subject to a bounded perturbation constraint:

$$\mathbf{x}_{\text{adv}} = \arg\max_{\mathbf{x}'} J(\boldsymbol{\theta}, \mathbf{x}', y) \quad \text{subject to} \quad \|\mathbf{x}' - \mathbf{x}\|_\infty \leq \epsilon. \tag{12.16}$$

Since computing the exact solution to this optimization problem is computationally expensive, FGSM employs a first-order approximation using Taylor series expansion:

$$J(\boldsymbol{\theta}, \mathbf{x} + \boldsymbol{\delta}, y) \approx J(\boldsymbol{\theta}, \mathbf{x}, y) + \boldsymbol{\delta}^T \nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y). \tag{12.17}$$

Maximizing this expression with respect to $\boldsymbol{\delta}$ under the ℓ_∞-norm constraint $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ leads to the optimal perturbation:

$$\boldsymbol{\delta}^* = \epsilon \, \mathrm{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)). \tag{12.18}$$

Substituting this into the perturbed input yields the FGSM formulation:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \, \mathrm{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)). \tag{12.19}$$

FGSM exploits the local linearity property of neural networks, which can be seen by considering a first-order approximation of the neural network output near $\mathbf{x}$:

$$f(\mathbf{x} + \boldsymbol{\delta}; \boldsymbol{\theta}) \approx f(\mathbf{x}; \boldsymbol{\theta}) + \nabla_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{\theta})^T \boldsymbol{\delta}. \tag{12.20}$$

Given that neural networks often have high-dimensional input spaces, even a small perturbation aligned with the gradient can induce significant changes in the output, leading to misclassification. If the network assigns class probabilities using a softmax function

$$P(y \mid \mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(f_y(\mathbf{x}; \boldsymbol{\theta}))}{\sum_j \exp(f_j(\mathbf{x}; \boldsymbol{\theta}))}, \tag{12.21}$$

then the FGSM attack perturbs $\mathbf{x}$ in a way that increases the probability of an incorrect class y', where

$$P(y' \mid \mathbf{x}_{\text{adv}}; \boldsymbol{\theta}) > P(y \mid \mathbf{x}_{\text{adv}}; \boldsymbol{\theta}). \tag{12.22}$$

The effectiveness of FGSM depends on ϵ. A small ϵ may not cause misclassification, whereas a large ϵ may introduce perceptible distortions. The impact of FGSM is often analyzed using the decision boundary properties of neural networks. Consider a linear classifier with decision boundary defined by

$$\mathbf{w}^T \mathbf{x} + b = 0. \tag{12.23}$$

Applying FGSM perturbs $\mathbf{x}$ along the gradient direction, which in a linear setting is simply

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \, \mathrm{sign}(\mathbf{w}). \tag{12.24}$$

If $\mathbf{x}$ is initially correctly classified, then after perturbation, the new decision function evaluation is

$$\mathbf{w}^T (\mathbf{x} + \epsilon \, \mathrm{sign}(\mathbf{w})) + b = \mathbf{w}^T \mathbf{x} + b + \epsilon \|\mathbf{w}\|_1. \tag{12.25}$$

If the perturbation shifts this beyond zero, the classification flips. This explains why FGSM can be highly effective, even for small ϵ, particularly in high-dimensional spaces where each small perturbation accumulates across dimensions. A key property of FGSM is its transferability across different models. Given two classifiers $f_1(\mathbf{x}; \boldsymbol{\theta}_1)$ and $f_2(\mathbf{x}; \boldsymbol{\theta}_2)$, adversarial examples crafted for f_1 often remain adversarial for f_2, which can be analyzed using the gradient similarity measure:

$$\mathbb{E}\left[\nabla_{\mathbf{x}} J_1(\boldsymbol{\theta}_1, \mathbf{x}, y) \cdot \nabla_{\mathbf{x}} J_2(\boldsymbol{\theta}_2, \mathbf{x}, y)\right] > 0. \tag{12.26}$$

This suggests that gradients of different models are often aligned, leading to the observed transferability phenomenon. FGSM can be countered using adversarial training, where the training objective is modified to incorporate adversarial examples:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} \left[\max_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} J(\boldsymbol{\theta}, \mathbf{x} + \boldsymbol{\delta}, y) \right]. \tag{12.27}$$

This formulation leads to robust classifiers that are less sensitive to adversarial perturbations. However, adversarial training increases computational costs, as it requires solving an inner maximization problem during training. Another defense mechanism is gradient masking, where modifications to the loss function result in non-informative gradients:

$$\tilde{J}(\boldsymbol{\theta}, \mathbf{x}, y) = J(\boldsymbol{\theta}, \mathbf{x}, y) + \lambda \|\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)\|^2. \tag{12.28}$$

While this suppresses adversarial perturbations, it does not fundamentally resolve the adversarial vulnerability, as adaptive attacks can circumvent such defenses by using alternative optimization techniques.

12.2 Projected Gradient Descent (PGD) method in Adversial Learning

Projected Gradient Descent (PGD) is a highly rigorous iterative optimization method employed in adversarial learning to generate adversarial examples that maximize the loss function of a given classifier while ensuring that the perturbation remains within a specified constraint set. The mathematical formulation of PGD follows directly from constrained optimization, where an adversarial perturbation $\boldsymbol{\delta}$ is sought such that a perturbed input $\boldsymbol{x} + \boldsymbol{\delta}$ induces a maximal classification error while remaining within a bounded perturbation set $\mathcal{S}$, often defined by an ℓ_p-norm ball. The adversarial attack is thus formalized as

$$\max_{\boldsymbol{\delta} \in \mathcal{S}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}) \tag{12.29}$$

where $\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})$ denotes the loss function (e.g., cross-entropy loss), $\boldsymbol{x}$ is the input sample, y is the corresponding label, and $\boldsymbol{\theta}$ represents the parameters of the model. The perturbation set $\mathcal{S}$ is commonly chosen as the ℓ_p-ball of radius ϵ, given by

$$\mathcal{S} = \{\boldsymbol{\delta} \in \mathbb{R}^d \mid \|\boldsymbol{\delta}\|_p \leq \epsilon\}. \tag{12.30}$$

To solve the constrained optimization problem, PGD proceeds iteratively by updating $\boldsymbol{\delta}$ in the direction of the gradient of $\mathcal{L}$ with respect to the input, followed by a projection step to enforce the constraint. Given a step size α, the update rule for PGD at iteration t is

$$\boldsymbol{\delta}^{(t+1)} = \Pi_{\mathcal{S}} \left(\boldsymbol{\delta}^{(t)} + \alpha \frac{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}^{(t)}, y; \boldsymbol{\theta})}{\|\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}^{(t)}, y; \boldsymbol{\theta})\|_p} \right), \tag{12.31}$$

where $\Pi_{\mathcal{S}}(\cdot)$ denotes the projection operator that ensures $\boldsymbol{\delta}^{(t+1)}$ remains within $\mathcal{S}$. The projection depends on the choice of the ℓ_p-norm constraint. For the commonly used ℓ_∞-ball, the projection is simply

$$\Pi_{\mathcal{S}}(\boldsymbol{\delta}) = \max(-\epsilon, \min(\boldsymbol{\delta}, \epsilon)), \tag{12.32}$$

which clips each component of $\boldsymbol{\delta}$ to lie within $[-\epsilon, \epsilon]$. For the ℓ_2-ball, projection is achieved by normalizing and scaling the perturbation as

$$\Pi_{\mathcal{S}}(\boldsymbol{\delta}) = \epsilon \frac{\boldsymbol{\delta}}{\|\boldsymbol{\delta}\|_2} \quad \text{if } \|\boldsymbol{\delta}\|_2 > \epsilon. \tag{12.33}$$

The iterative application of this process ensures that the perturbation maximally increases the loss while adhering to the predefined perturbation budget. In the limit as the step size $\alpha \to 0$ and the number of iterations $T \to \infty$, PGD approximates the optimal solution of the constrained maximization problem. This is in contrast to the Fast Gradient Sign Method (FGSM), which performs only a single-step update of the form

$$\boldsymbol{\delta} = \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})\right). \tag{12.34}$$

PGD is therefore considered a stronger attack, as it allows the adversarial perturbation to explore the loss landscape more effectively by iteratively refining the adversarial example. From a theoretical perspective, PGD can be interpreted as an approximate projected gradient ascent method for solving the constrained maximization problem. Formally, this corresponds to a Lagrangian formulation where the Karush-Kuhn-Tucker (KKT) conditions dictate the optimal perturbation:

$$\nabla_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}) - \lambda \nabla_{\boldsymbol{\delta}} g(\boldsymbol{\delta}) = 0, \tag{12.35}$$

where $g(\boldsymbol{\delta})$ defines the boundary of the perturbation set. The iterative updates in PGD approximate this equilibrium condition numerically. Moreover, when applied iteratively with different random initializations of $\boldsymbol{\delta}^{(0)}$, PGD can better explore non-convex loss surfaces, making it an effective strategy against adversarially trained defenses.

12.3 Generative approach in Adversarial Learning

The generative approach in adversarial learning is fundamentally rooted in the interplay between two networks: the generator G and the discriminator D, which are trained in an adversarial manner. The generator aims to synthesize data that is indistinguishable from real data, while the discriminator attempts to distinguish real data from generated data. This setup can be rigorously modeled as a minimax optimization problem, given by

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{12.36}$$

where p_{data} represents the true data distribution, and $p_{\mathbf{z}}$ is the prior distribution from which latent variables $\mathbf{z}$ are drawn. The generator learns a mapping $G : \mathbf{z} \to \mathbf{x}$, parameterized by θ_G, such that the distribution of generated samples $p_G(\mathbf{x})$ approximates p_{data}. The discriminator, parameterized by θ_D, learns a function $D : \mathbf{x} \to [0, 1]$, which estimates the probability that $\mathbf{x}$ is drawn from p_{data}. The training of the generator and discriminator proceeds iteratively, where D is updated by maximizing

$$J_D(\theta_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{12.37}$$

and G is updated by minimizing

$$J_G(\theta_G) = \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{12.38}$$

A fundamental result in adversarial learning is that, given sufficient capacity, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \tag{12.39}$$

Substituting D^* into the minimax objective leads to the Jensen-Shannon divergence between p_{data} and p_G, given by

$$\text{JSD}(p_{\text{data}} \parallel p_G) = \frac{1}{2} D_{\text{KL}}(p_{\text{data}} \parallel M) + \frac{1}{2} D_{\text{KL}}(p_G \parallel M) \tag{12.40}$$

where

$$M = \frac{1}{2}(p_{\text{data}} + p_G) \tag{12.41}$$

and $D_{\text{KL}}(P \parallel Q)$ denotes the Kullback-Leibler divergence

$$D_{\text{KL}}(P \parallel Q) = \int P(\mathbf{x}) \log \frac{P(\mathbf{x})}{Q(\mathbf{x})} \, d\mathbf{x} \tag{12.42}$$

In practical settings, this divergence-based formulation leads to issues such as vanishing gradients when p_G and p_{data} do not overlap significantly. To address this, alternative formulations such as Wasserstein GANs (WGANs) optimize the Wasserstein distance

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \le 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_G}[f(\mathbf{x})] \tag{12.43}$$

where f is constrained to be 1-Lipschitz. The Wasserstein distance provides meaningful gradients even when p_G and p_{data} have disjoint support. The optimal transport formulation of the Wasserstein distance can be rewritten as

$$W(p_{\text{data}}, p_G) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_G)} \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \gamma}[\|\mathbf{x} - \mathbf{y}\|] \tag{12.44}$$

where $\Pi(p_{\text{data}}, p_G)$ denotes the set of joint distributions with marginals p_{data} and p_G. A crucial extension of adversarial learning is its application to conditional generation, where the generator learns a conditional distribution $p_G(\mathbf{x} \mid \mathbf{y})$. The objective function for a conditional GAN is given by

$$\min_G \max_D \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim p_{\text{data}}}[\log D(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}, \mathbf{y} \sim p_{\text{data}}}[\log(1 - D(G(\mathbf{z}, \mathbf{y}), \mathbf{y}))] \tag{12.45}$$

From a probabilistic standpoint, generative adversarial networks can be interpreted within the framework of energy-based models, where the discriminator implicitly defines an energy function

$$E(\mathbf{x}) = -\log D(\mathbf{x}) \tag{12.46}$$

Minimizing this energy aligns generated samples with the modes of the true distribution, reinforcing the generative modeling capability of adversarial learning.

12.4 Interpreting Generative Adversarial Networks within the framework of Energy-Based models

Generative Adversarial Networks (GANs) can be interpreted probabilistically within the framework of Energy-Based Models (EBMs) by considering their fundamental objective as a minimax optimization problem that implicitly defines an energy function governing the probability distribution of data and generated samples. The generator G and the discriminator D interact in a two-player game, which can be rigorously formulated in the language of probabilistic energy-based models. The discriminator learns to distinguish real data samples from generated ones, while the generator learns to produce samples that minimize the discriminator's ability to differentiate them.

To formally understand this, let $p_{\text{data}}(\mathbf{x})$ be the true data distribution and $p_G(\mathbf{x})$ be the generator's induced distribution. The discriminator is parameterized by $D_\theta(\mathbf{x})$, where $D_\theta(\mathbf{x})$ outputs the probability that $\mathbf{x}$ is a real sample. The optimal discriminator in the standard GAN formulation is derived as:

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \tag{12.47}$$

which emerges from the minimization of the following objective:

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_G}[\log(1 - D(\mathbf{x}))] \tag{12.48}$$

In the energy-based interpretation, we introduce an energy function $E(\mathbf{x})$ that models the relative likelihood of a sample. The energy function is linked to probability distributions via the Gibbs measure:

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{12.49}$$

where Z is the partition function:

$$Z = \int e^{-E(\mathbf{x})} d\mathbf{x} \tag{12.50}$$

We can define an energy-based discriminator as:

$$D_\theta(\mathbf{x}) = \sigma(-E_\theta(\mathbf{x})) = \frac{1}{1 + e^{E_\theta(\mathbf{x})}} \tag{12.51}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. This formulation implies that minimizing the GAN objective is equivalent to learning an energy function $E_\theta(\mathbf{x})$ such that:

$$E_\theta(\mathbf{x}) \approx -\log \frac{p_{\text{data}}(\mathbf{x})}{p_G(\mathbf{x})} \tag{12.52}$$

which is precisely the log-ratio of the real and generated distributions, making GANs inherently an instance of energy-based models. From a probabilistic standpoint, the generator learns to minimize the divergence between $p_G(\mathbf{x})$ and $p_{\text{data}}(\mathbf{x})$. Specifically, for a fixed discriminator, the generator's objective can be rewritten as minimizing the Jensen-Shannon divergence:

$$D_{\text{JS}}(p_{\text{data}} \parallel p_G) = \frac{1}{2} D_{\text{KL}}(p_{\text{data}} \parallel M) + \frac{1}{2} D_{\text{KL}}(p_G \parallel M) \tag{12.53}$$

where $M = \frac{1}{2}(p_{\text{data}} + p_G)$. This reveals that GANs operate within an information-theoretic framework where they reduce an energy-based measure of dissimilarity between real and generated samples. The training dynamics can be interpreted through contrastive divergence, which appears naturally in energy-based models. The generator updates its parameters to produce samples that have lower energy, thereby implicitly performing a form of Markov Chain Monte Carlo (MCMC)-like sampling in the energy landscape defined by $E_\theta(\mathbf{x})$. The gradient update of the generator is:

$$\nabla_{\theta_G} \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))] = -\mathbb{E}_{\mathbf{z}\sim p_z}\left[\nabla_{\theta_G} E_\theta(G(\mathbf{z}))\right] \tag{12.54}$$

which aligns with the contrastive divergence updates used in training energy-based models. This equivalence suggests that GANs effectively perform energy minimization by dynamically adjusting G such that its samples reduce the energy discrepancy between real and generated distributions. Furthermore, considering a continuous-time stochastic formulation, the discriminator can be reinterpreted using a Langevin-type evolution:

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) + \sqrt{2\beta^{-1}}\eta(t) \tag{12.55}$$

where $\eta(t)$ is a white noise process and β controls the temperature of the energy model. This perspective suggests that adversarial training implicitly simulates diffusion in an energy-based potential field, refining the generator's distribution p_G over time. Thus, from a probabilistic viewpoint, generative adversarial networks fundamentally operate as an energy-based model where the discriminator estimates an energy function that guides the generator toward producing samples that match the statistics of real data. This establishes GANs as a probabilistic framework akin to energy minimization methods such as Boltzmann Machines, where adversarial training serves as an implicit mechanism for defining an energy landscape in the data space.

13 Casual Inference in Deep Neural Networks

Causal inference in deep neural networks involves the rigorous mathematical study of cause-and-effect relationships within high-dimensional representations learned by deep models. Unlike traditional statistical correlation, causal inference seeks to understand how perturbations in input variables propagate through the model to affect outcomes, disentangling spurious correlations from true causation. Given a set of random variables X, Y, and Z, a fundamental question in causal inference is whether X causally influences Y or whether their observed association is due to a confounding factor Z. The structural causal model (SCM) framework formalizes this using a set of structural equations:

$$Y = f_Y(X, U_Y) \tag{13.1}$$

$$X = f_X(Z, U_X) \tag{13.2}$$

where f_Y and f_X are deterministic functions encoding the causal mechanisms, and U_Y and U_X are exogenous noise variables, assumed to be mutually independent. The central problem in causal inference is identifying causal effects through interventions, denoted as $do(X = x)$, which replace the original structural equation of X with a fixed value x, leading to a new distribution:

$$P(Y|do(X = x)) = \sum_z P(Y|X = x, Z = z)P(Z) \tag{13.3}$$

This contrasts with traditional conditional probabilities $P(Y|X)$, which do not eliminate confounding. In deep neural networks, causal inference is particularly challenging due to the high-dimensional, entangled nature of representations. Given a deep neural network parameterized by weights θ, mapping an input X to an output Y through multiple layers:

$$H^{(l)} = \sigma(W^{(l)}H^{(l-1)} + b^{(l)}) \tag{13.4}$$

where $H^{(l)}$ denotes the activations at layer l, $W^{(l)}$ and $b^{(l)}$ are the weight matrices and biases, and σ is a nonlinear activation function, causal inference seeks to identify whether perturbations in X affect Y via a causal pathway. One approach to formalizing causality in deep networks is the use of causal feature selection, which attempts to identify a subset of features S such that:

$$P(Y|do(X_S = x_S)) = P(Y|X_S = x_S) \tag{13.5}$$

for all x_S in the support of X_S, ensuring that selected features S are causally relevant to Y. This can be achieved using conditional independence tests based on the back-door criterion:

$$P(Y|do(X)) = \sum_Z P(Y|X, Z)P(Z) \tag{13.6}$$

where Z blocks all back-door paths between X and Y. In deep neural networks, this translates to modifying network architectures to impose structural constraints that enforce causal dependencies. A key tool in causal inference for deep learning is the concept of counterfactual reasoning. Given an observed instance $X = x$ leading to output $Y = y$, a counterfactual query asks what would have happened had X been different:

$$Y_{X \leftarrow x'} = f_Y(f_X^{-1}(x', U_X), U_Y) \tag{13.7}$$

where $X \leftarrow x'$ denotes an intervention setting X to x', and f_X^{-1} reconstructs the counterfactual world. Estimating counterfactuals in deep models requires generative approaches such as variational autoencoders (VAEs) and normalizing flows, which approximate the latent space U from which counterfactual samples can be drawn. The fundamental theorem of causality states that if a probability distribution $P(X, Y)$ is generated from an SCM, then all counterfactual quantities are identifiable from the structural equations. Given a trained neural network with loss function $\mathcal{L}(\theta)$, causal constraints can be imposed by regularizing for causal invariance:

$$\mathcal{L}(\theta) + \lambda \sum_{i=1}^{n} |P(Y|X_i) - P(Y|do(X_i))| \tag{13.8}$$

where λ controls the strength of the causal penalty. This ensures that deep models learn representations aligned with causal mechanisms rather than spurious correlations. Another crucial aspect is the causal transportability of learned models, ensuring that a model trained in one environment generalizes to another with different distributions $P(X)$. This is characterized by domain adaptation techniques that minimize the divergence:

$$D(P(Y|do(X)), P'(Y|do(X))) \tag{13.9}$$

where P' is the distribution in the target domain, and D is a divergence measure such as KL-divergence:

$$D_{\text{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{13.10}$$

In deep networks, this is achieved using domain-invariant representations through adversarial training or contrastive learning. A critical issue in causal inference is the identifiability of causal effects from observational data. The fundamental problem of causal inference states that counterfactual outcomes cannot be directly observed:

$$Y_{X \leftarrow x} \neq Y_{X \leftarrow x'} \tag{13.11}$$

for $x \neq x'$, meaning that estimating $P(Y|do(X))$ requires additional assumptions such as unconfoundedness:

$$Y_{X \leftarrow x} \perp\!\!\!\perp X|Z \tag{13.12}$$

which allows estimation using propensity score matching:

$$P(Y|do(X)) = \sum_Z P(Y|X, Z)P(Z) \tag{13.13}$$

Thus, deep causal inference seeks to disentangle causal structure within neural networks, ensuring that learned representations encode true causal effects rather than spurious correlations. This requires integrating structural causal models, counterfactual reasoning, domain adaptation, and invariant risk minimization, ensuring that deep models generalize to unseen domains while preserving causal mechanisms.

13.1 Structural Causal Model (SCM)

A **Structural Causal Model (SCM)** is a rigorous mathematical framework for representing and reasoning about causal relationships between variables. It formalizes causality using **structural**

equations and **directed acyclic graphs (DAGs)** to capture cause-and-effect mechanisms. Given a set of endogenous variables $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$ and exogenous variables $\mathbf{U} = \{U_1, U_2, \ldots, U_m\}$, an SCM is defined as a tuple $(\mathbf{V}, \mathbf{U}, \mathbf{F}, P(\mathbf{U}))$, where $\mathbf{F}$ is a set of deterministic structural equations, and $P(\mathbf{U})$ is a probability distribution over exogenous variables. Each endogenous variable V_i is determined by a function of its parents in the causal graph and the corresponding exogenous variables:

$$V_i = f_i(\mathrm{PA}_i, U_i) \tag{13.14}$$

where PA_i represents the set of parent variables of V_i in the causal graph. The causal relationships between variables are represented using a directed acyclic graph $G = (\mathbf{V}, E)$, where each edge $V_j \to V_i$ signifies that V_j is a direct cause of V_i. The absence of an edge between V_j and V_i indicates conditional independence given other parents. The structural equations define the causal mechanism, and their functional form encodes counterfactual relationships. Given an intervention $do(X = x)$, where the variable X is forcibly set to x, the new SCM modifies the equation for X while keeping all other equations unchanged:

$$SCM_{do(X=x)} = (\mathbf{V}, \mathbf{U}, \mathbf{F}^{X=x}, P(\mathbf{U})) \tag{13.15}$$

where $\mathbf{F}^{X=x}$ represents the modified set of structural equations with X replaced by the constant x. Causal effects are quantified using interventional distributions. The post-intervention distribution of an outcome Y given an intervention on X is computed using the **causal effect formula**:

$$P(Y \mid do(X = x)) = \sum_{\mathbf{V} \setminus \{X,Y\}} P(\mathbf{V} \setminus \{X\} \mid X = x) P(X = x) \tag{13.16}$$

which integrates out all non-intervened variables. The backdoor criterion provides a criterion for identifying $P(Y \mid do(X))$ when a set of variables $\mathbf{Z}$ blocks all backdoor paths from X to Y, leading to the adjustment formula:

$$P(Y \mid do(X)) = \sum_{\mathbf{Z}} P(Y \mid X, \mathbf{Z}) P(\mathbf{Z}) \tag{13.17}$$

Counterfactuals are evaluated by computing potential outcomes. Given an observed world where $X = x$ and $Y = y$, the counterfactual query $Y_{X=x'}$ seeks the value Y would have taken had X been set to x'. This is formalized using the **three-step counterfactual algorithm**:

1. **Abduction:** Infer exogenous variables using the observed data:

$$U = f_X^{-1}(X) \tag{13.18}$$

2. **Action:** Modify the SCM by replacing X with x':

$$SCM_{do(X=x')} \tag{13.19}$$

3. **Prediction:** Solve for $Y_{X=x'}$ in the modified SCM:

$$Y_{X=x'} = f_Y(\mathrm{PA}_Y, U_Y) \tag{13.20}$$

The probability of causation (PC) quantifies the likelihood that X caused Y by comparing the counterfactual and factual outcomes:

$$PC = P(Y_{X=1} = 1 \mid X = 0, Y = 0) \tag{13.21}$$

which requires structural knowledge of the system. Identification of causal effects depends on graphical conditions such as the **front-door criterion**, which allows inference of $P(Y \mid do(X))$ even when confounding is unobserved, provided an intermediate variable M satisfies:

$$P(Y \mid do(X)) = \sum_{m} P(Y \mid m) P(m \mid do(X)) \tag{13.22}$$

SCMs generalize probabilistic models by encoding mechanistic relationships rather than mere statistical associations. The key distinction is that SCMs allow answering counterfactual queries, distinguishing correlation from causation, and supporting robust decision-making under interventions.

13.2 Counterfactual reasoning in causal inference for deep neural networks

Counterfactual reasoning in causal inference for deep neural networks is fundamentally grounded in the potential outcomes framework. Suppose we define an observed data distribution as $\mathcal{D} = \{(\mathbf{x}_i, t_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the input features, $t_i \in \{0, 1\}$ denotes the binary treatment assignment, and $y_i \in \mathbb{R}$ represents the observed outcome. In the potential outcomes framework, each unit i has two possible outcomes: $Y_i(1)$ when treated and $Y_i(0)$ when untreated. The observed outcome is expressed as

$$Y_i = t_i Y_i(1) + (1 - t_i) Y_i(0). \tag{13.23}$$

However, the counterfactual outcome, which is the outcome corresponding to the treatment level not received, remains unobserved. The fundamental problem of causal inference lies in estimating this counterfactual quantity. The treatment effect for an individual i is given by

$$\tau_i = Y_i(1) - Y_i(0), \tag{13.24}$$

but due to the missing counterfactual, it is not directly observable. The average treatment effect (ATE) is expressed as

$$\mathbb{E}[\tau] = \mathbb{E}[Y(1) - Y(0)], \tag{13.25}$$

which requires estimation of both the factual and counterfactual expectations. One approach to estimating counterfactuals in deep neural networks is using representation learning to balance treatment groups. A neural network is trained to learn a latent representation $\Phi(\mathbf{x})$ such that the distributions of treated and untreated units become similar:

$$\Phi(\mathbf{x}) = f_\theta(\mathbf{x}), \tag{13.26}$$

where f_θ is a deep neural network parameterized by θ. The objective function often incorporates a domain discrepancy measure such as the Integral Probability Metric (IPM):

$$\mathcal{L}_{\text{IPM}} = \sup_{h \in \mathcal{H}} \left| \mathbb{E}_{p(\Phi(\mathbf{x})|T=1)}[h(\Phi(\mathbf{x}))] - \mathbb{E}_{p(\Phi(\mathbf{x})|T=0)}[h(\Phi(\mathbf{x}))] \right|. \tag{13.27}$$

By minimizing this term, the network ensures that the latent distributions for treated and control groups are similar, facilitating counterfactual inference. A common loss function for training the neural network includes both factual prediction loss and a regularization term for domain alignment:

$$\mathcal{L} = \sum_{i=1}^N \ell(Y_i, \hat{Y}_i) + \lambda \mathcal{L}_{\text{IPM}}. \tag{13.28}$$

A counterfactual prediction $\hat{Y}_i(1 - t_i)$ is obtained by passing $\mathbf{x}_i$ through the network trained on both factual and imputed counterfactual distributions:

$$\hat{Y}_i(1) = g_\theta(\Phi(\mathbf{x}_i), 1), \quad \hat{Y}_i(0) = g_\theta(\Phi(\mathbf{x}_i), 0), \tag{13.29}$$

where g_θ is the output layer of the neural network parameterized by θ. To improve counterfactual estimation, techniques such as adversarial balancing loss and domain adaptation methods may be incorporated:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{p(\Phi(\mathbf{x})|T=1)}[\log D(\Phi(\mathbf{x}))] + \mathbb{E}_{p(\Phi(\mathbf{x})|T=0)}[\log(1 - D(\Phi(\mathbf{x})))]. \tag{13.30}$$

Here, $D(\Phi(\mathbf{x}))$ is a discriminator network that tries to distinguish between treated and untreated units in latent space, and its adversarial training ensures treatment group similarity. Thus, counterfactual reasoning in deep neural networks is achieved by leveraging representation learning, adversarial domain adaptation, and balanced predictive modeling to estimate missing potential outcomes.

13.3 Domain adaptation in causal inference within deep neural networks

Domain adaptation in causal inference within deep neural networks is fundamentally concerned with the transfer of knowledge from a source domain $\mathcal{S}$ to a target domain $\mathcal{T}$, where the distributions of the input features and the causal mechanisms governing the relationships among variables differ between domains. Given an input space $\mathcal{X}$, a covariate space $\mathcal{C}$, an outcome space $\mathcal{Y}$, and a treatment space $\mathcal{A}$, we consider the problem of estimating the causal effect of treatment $A \in \mathcal{A}$ on outcome $Y \in \mathcal{Y}$ under domain shift. Specifically, let $P_{\mathcal{S}}(X, A, Y)$ and $P_{\mathcal{T}}(X, A, Y)$ denote the joint distributions in the source and target domains, respectively. The key assumption in causal domain adaptation is that while the distribution of covariates $P_{\mathcal{S}}(X)$ and $P_{\mathcal{T}}(X)$ may differ, the underlying structural causal model (SCM) governing the treatment and outcome relationship remains invariant across domains. Mathematically, the invariance of causal mechanisms implies that the conditional distributions of treatment assignment and outcome given the covariates are domain-independent:

$$P_{\mathcal{S}}(A \mid X) = P_{\mathcal{T}}(A \mid X), \quad P_{\mathcal{S}}(Y \mid X, A) = P_{\mathcal{T}}(Y \mid X, A). \tag{13.31}$$

However, direct inference of the causal effect in the target domain is hindered by the discrepancy between the marginal distributions $P_{\mathcal{S}}(X)$ and $P_{\mathcal{T}}(X)$, a phenomenon known as covariate shift. This necessitates a reweighting strategy to correct for the distributional shift. A common approach involves the use of importance weights:

$$w(X) = \frac{P_{\mathcal{T}}(X)}{P_{\mathcal{S}}(X)}, \tag{13.32}$$

which can be estimated via density ratio estimation techniques such as kernel mean matching or adversarial learning. Given these weights, the expectation of any function $f(X, A, Y)$ in the target domain can be approximated using observations from the source domain:

$$\mathbb{E}_{\mathcal{T}}[f(X, A, Y)] \approx \mathbb{E}_{\mathcal{S}}\left[w(X) f(X, A, Y)\right]. \tag{13.33}$$

A deep neural network-based approach to causal domain adaptation typically parameterizes the outcome model $h_\theta(X, A)$ as a neural network with parameters θ, trained to minimize the weighted empirical loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n_{\mathcal{S}}} w(X_i) \ell(h_\theta(X_i, A_i), Y_i), \tag{13.34}$$

where $\ell(\cdot, \cdot)$ is a suitable loss function such as mean squared error for continuous outcomes or cross-entropy loss for binary outcomes. Additionally, domain-adversarial neural networks (DANNs) introduce a domain discriminator $D_\phi(X)$, parameterized by ϕ, trained to distinguish between source and target samples while the feature extractor $g_\theta(X)$ is trained to minimize the discriminator's ability to differentiate domains, effectively promoting invariant feature representations:

$$\min_{\theta} \max_{\phi} \sum_{i=1}^{n_{\mathcal{S}}} \log D_\phi(g_\theta(X_i)) + \sum_{j=1}^{n_{\mathcal{T}}} \log(1 - D_\phi(g_\theta(X_j))). \tag{13.35}$$

By integrating the domain-invariant representation $g_\theta(X)$ with causal inference models, the confounding bias induced by distribution shift is mitigated, allowing for unbiased estimation of the causal effect in the target domain. Furthermore, the potential outcome framework in deep learning-based causal domain adaptation estimates the counterfactual outcomes Y^0, Y^1 using a shared feature encoder $f_\theta(X)$ and domain-invariant treatment-response functions $h_{\theta_0}(X)$ and $h_{\theta_1}(X)$:

$$\hat{Y}^0 = h_{\theta_0}(f_\theta(X)), \quad \hat{Y}^1 = h_{\theta_1}(f_\theta(X)). \tag{13.36}$$

The individualized treatment effect (ITE) is then estimated as:

$$\hat{\tau}(X) = \hat{Y}^1 - \hat{Y}^0. \tag{13.37}$$

To ensure domain generalization, recent advances incorporate representation learning techniques such as contrastive learning and information bottlenecks, where the feature extractor $g_\theta(X)$ is regularized to minimize the mutual information between domain-specific variations and the learned representations:

$$\min_\theta \mathbb{E}_{X \sim P_{\mathcal{S}}(X) \cup P_{\mathcal{T}}(X)} \left[\|g_\theta(X) - g_\theta(\tilde{X})\|_2^2 \right], \tag{13.38}$$

where $\tilde{X}$ represents an augmented version of X through domain-specific transformations. The overall objective function in deep causal domain adaptation thus integrates causal loss, domain invariance constraints, and representation learning regularization:

$$\min_{\theta,\theta_0,\theta_1} \sum_{i=1}^{n_{\mathcal{S}}} w(X_i)\ell(h_{\theta_{A_i}}(f_\theta(X_i)), Y_i) + \lambda \sum_{j=1}^{n_{\mathcal{T}}} \log(1 - D_\phi(g_\theta(X_j))) + \beta \mathbb{E}_{X,\tilde{X}} \left[\|g_\theta(X) - g_\theta(\tilde{X})\|_2^2 \right], \tag{13.39}$$

where λ and β are hyperparameters controlling the trade-off between causal effect estimation, domain adaptation, and representation alignment. By leveraging deep learning-based approaches for domain adaptation in causal inference, robust estimation of treatment effects is achieved even in the presence of significant distributional shifts, thereby enhancing the reliability of causal effect estimation in real-world applications.

13.4 Invariant Risk Minimization (IRM) in causal inference for deep neural networks

Invariant Risk Minimization (IRM) is a paradigm in causal inference and deep learning that aims to learn representations of data that lead to predictors that are invariant across different environments. Consider a dataset collected from multiple environments $\mathcal{E}$, each defined as a probability distribution $P_e(X, Y)$, where X denotes the input features, and Y represents the target variable. The goal of IRM is to learn a predictor $f : X \to Y$ that maintains its predictive performance across all possible environments, thereby capturing the causal relationship between X and Y rather than spurious correlations specific to particular environments.

Mathematically, let $\mathcal{H}$ denote the hypothesis space, which includes functions h parameterized by a deep neural network. We define an invariant predictor h that minimizes the worst-case risk over all observed environments:

$$\min_{h \in \mathcal{H}} \max_{e \in \mathcal{E}} R_e(h), \tag{13.40}$$

where the empirical risk in environment e is given by:

$$R_e(h) = \mathbb{E}_{(X,Y)\sim P_e}[\ell(h(X), Y)], \tag{13.41}$$

with ℓ being a loss function, commonly the squared error loss for regression:

$$\ell(h(X), Y) = (h(X) - Y)^2, \tag{13.42}$$

or the cross-entropy loss for classification:

$$\ell(h(X), Y) = -Y \log h(X) - (1 - Y) \log(1 - h(X)). \tag{13.43}$$

A crucial observation is that empirical risk minimization (ERM), which minimizes the sum of risks over all environments,

$$\min_{h \in \mathcal{H}} \sum_{e \in \mathcal{E}} R_e(h), \tag{13.44}$$

often leads to predictors that exploit spurious correlations present in specific environments rather than capturing the invariant relationship. To address this, IRM introduces an additional constraint to enforce invariance in the predictor. Specifically, it seeks a representation function $\Phi : X \to Z$ such that a classifier w trained on Z remains optimal across all environments:

$$\Phi^*, w^* = \arg\min_{\Phi, w} \sum_{e \in \mathcal{E}} R_e(w \circ \Phi), \tag{13.45}$$

subject to the invariance constraint:

$$w \in \arg\min_{\tilde{w}} R_e(\tilde{w} \circ \Phi) \quad \forall e \in \mathcal{E}. \tag{13.46}$$

To relax this constraint into a differentiable objective, the IRM penalty is introduced:

$$\mathcal{L}_{\text{IRM}} = \sum_{e \in \mathcal{E}} R_e(w \circ \Phi) + \lambda \|\nabla_w R_e(w \circ \Phi)|_{w=1}\|^2. \tag{13.47}$$

Here, the gradient term ensures that the classifier w remains optimal across all environments by enforcing that its gradient with respect to w vanishes. The hyperparameter λ controls the strength of this regularization. Expanding the penalty term,

$$\|\nabla_w R_e(w \circ \Phi)|_{w=1}\|^2 = \sum_{e \in \mathcal{E}} \left\| \mathbb{E}_{(X,Y) \sim P_e} \left[\nabla_w \ell(w \Phi(X), Y) \right] \right\|^2, \tag{13.48}$$

ensuring that the gradients of the risks are aligned across environments. A deeper connection to causality emerges when considering the Structural Equation Model (SEM):

$$Y = f^*(X) + \epsilon, \quad X = g(Z, S), \tag{13.49}$$

where Z are the causal features, S are spurious features, and ϵ is an independent noise term. An ERM-trained deep network might learn S because it improves empirical risk minimization in observed environments, while IRM discourages dependence on S by enforcing that the predictor generalizes across all environments. A theoretical justification for IRM follows from an analysis of the optimal invariant predictor. Define the optimal classifier for a given representation Φ:

$$w_e^* = \arg\min_{w} R_e(w \circ \Phi). \tag{13.50}$$

For a truly invariant representation Φ^*, the classifier should be identical across all environments:

$$w_e^* = w^* \quad \forall e \in \mathcal{E}. \tag{13.51}$$

This leads to the condition:

$$\nabla_w R_e(w \circ \Phi) = 0 \quad \forall e \in \mathcal{E}, \tag{13.52}$$

which is precisely the IRM penalty. Empirically, training with IRM often results in improved out-of-distribution generalization compared to standard deep learning approaches. Finally, an alternative interpretation of IRM emerges from considering a variational bound on the mutual information between the learned representation and the causal variables:

$$I(Z; Y) = H(Y) - H(Y|Z). \tag{13.53}$$

IRM implicitly maximizes this information while minimizing the conditional entropy $H(Y|Z)$, leading to representations that capture causal relationships rather than spurious correlations.

13.5 Empirical Risk Minimization (ERM) in causal inference for deep neural networks

Empirical Risk Minimization (ERM) is a fundamental principle in statistical learning theory, providing a framework for optimizing predictive models, particularly within deep neural networks (DNNs). In the context of causal inference, ERM is adapted to account for the complexities introduced by confounding variables, treatment assignment mechanisms, and counterfactual estimation. Given a dataset $\mathcal{D} = \{(X_i, T_i, Y_i)\}_{i=1}^N$, where $X_i \in \mathbb{R}^d$ represents covariates, $T_i \in \{0, 1\}$ denotes the binary treatment assignment, and $Y_i \in \mathbb{R}$ is the observed outcome, the goal of causal inference is to estimate the Individual Treatment Effect (ITE), defined as

$$\tau(X) = \mathbb{E}[Y \mid X, T = 1] - \mathbb{E}[Y \mid X, T = 0]. \tag{13.54}$$

The empirical risk in classical supervised learning is given by

$$\hat{R}(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(X_i), Y_i), \tag{13.55}$$

where $h : \mathbb{R}^d \to \mathbb{R}$ is the hypothesis function parameterized by a deep neural network and $\ell(\cdot, \cdot)$ is a loss function, such as the squared loss

$$\ell(h(X_i), Y_i) = (h(X_i) - Y_i)^2. \tag{13.56}$$

However, in causal inference, the presence of treatment assignment T necessitates a modified objective that accounts for the potential outcome framework. A common approach is to estimate the potential outcomes $Y(0)$ and $Y(1)$ using separate deep neural networks, $h_0(X; \theta_0)$ and $h_1(X; \theta_1)$, parameterized by θ_0, θ_1, leading to the empirical risk formulation

$$\hat{R}(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \Big(T_i \ell(h_1(X_i; \theta_1), Y_i) + (1 - T_i) \ell(h_0(X_i; \theta_0), Y_i) \Big). \tag{13.57}$$

This objective function is a weighted empirical risk based on the observed treatment assignment, ensuring that only the factual outcome is used in training. The challenge arises due to the imbalance in treatment assignments, leading to potential covariate shift between the treated and control groups. To mitigate this, inverse propensity weighting (IPW) is often incorporated, where the empirical risk is modified as

$$\hat{R}_{\text{IPW}}(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{T_i}{e(X_i)} \ell(h_1(X_i; \theta_1), Y_i) + \frac{1 - T_i}{1 - e(X_i)} \ell(h_0(X_i; \theta_0), Y_i) \right), \tag{13.58}$$

where $e(X) = P(T = 1 \mid X)$ is the propensity score. This adjustment ensures that the empirical distribution of covariates in both treatment groups better approximates the underlying population distribution. However, direct inverse weighting can lead to high variance, motivating the use of balancing representations via domain adaptation methods such as representation learning with deep neural networks. Specifically, a feature representation function $\Phi : \mathbb{R}^d \to \mathbb{R}^m$ is learned such that the treated and control groups become indistinguishable in the transformed space. The empirical risk in this case is

$$\hat{R}_{\Phi}(\theta_0, \theta_1, \phi) = \frac{1}{N} \sum_{i=1}^{N} \Big(T_i \ell(h_1(\Phi(X_i; \phi); \theta_1), Y_i) + (1 - T_i) \ell(h_0(\Phi(X_i; \phi); \theta_0), Y_i) \Big), \tag{13.59}$$

with an additional discrepancy term penalizing differences between the distributions of $\Phi(X)$ under $T = 1$ and $T = 0$, such as the Maximum Mean Discrepancy (MMD),

$$\mathcal{D}(\Phi) = \left\| \frac{1}{N_T} \sum_{i:T_i=1} k(\Phi(X_i), \cdot) - \frac{1}{N_C} \sum_{i:T_i=0} k(\Phi(X_i), \cdot) \right\|_{\mathcal{H}}, \tag{13.60}$$

where $k(\cdot, \cdot)$ is a kernel function and $\mathcal{H}$ is the reproducing kernel Hilbert space. The final objective function then combines empirical risk minimization with regularization for balancing:

$$\min_{\theta_0, \theta_1, \phi} \hat{R}_\Phi(\theta_0, \theta_1, \phi) + \lambda \mathcal{D}(\Phi). \tag{13.61}$$

An alternative regularization strategy is adversarial balancing, where a discriminator $D(X)$ is trained to distinguish between treated and control units, and $\Phi(X)$ is trained adversarially to make this discrimination difficult. The objective for Φ and D is formulated as a min-max problem:

$$\min_{\theta_0, \theta_1, \phi} \max_D \hat{R}_\Phi(\theta_0, \theta_1, \phi) - \lambda \sum_{i=1}^{N} \log D(\Phi(X_i))^{T_i} (1 - D(\Phi(X_i)))^{1-T_i}. \tag{13.62}$$

This ensures that the learned representation is treatment-agnostic, improving generalizability to counterfactual estimation. Additionally, doubly robust methods integrate both outcome modeling and inverse weighting by introducing a correction term based on residuals from a separate outcome model $m(X)$:

$$\hat{R}_{\mathrm{DR}}(\theta_0, \theta_1, \phi) = \frac{1}{N} \sum_{i=1}^{N} \left(\ell(h_{T_i}(\Phi(X_i; \phi); \theta_{T_i}), Y_i) + \frac{T_i - e(X_i)}{e(X_i)(1 - e(X_i))} (Y_i - m(X_i)) \right). \tag{13.63}$$

This formulation ensures consistency even when either the outcome model or the propensity score model is misspecified. Ultimately, the optimization of these empirical risk formulations in deep neural networks involves gradient-based methods, with stochastic gradient descent (SGD) or Adam optimizing θ_0, θ_1, ϕ, while the propensity score model $e(X)$ is learned via logistic regression or a separate neural network. The complexity of ERM in causal deep learning thus lies in balancing factual accuracy, counterfactual generalization, and representation learning to ensure robust ITE estimation.

14 Network Architecture Search (NAS) in Deep Neural Networks

Network Architecture Search (NAS) in Deep Neural Networks (DNNs) is an optimization problem that seeks to automatically design the best neural network architecture for a given task, typically involving a search space $\mathcal{A}$, a search strategy $\mathcal{S}$, and a performance evaluation metric $\mathcal{E}$. Mathematically, the NAS problem can be framed as finding the optimal architecture α^* that maximizes a performance function $\mathcal{F}$, which can be formally written as

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathcal{F}(\alpha, \mathcal{D}) \tag{14.1}$$

where $\mathcal{D}$ represents the dataset used for training and evaluation. The search space $\mathcal{A}$ defines the possible neural network architectures, which can be represented as a directed acyclic graph (DAG) $G = (V, E)$, where nodes V correspond to layers (such as convolutional, fully connected, or recurrent layers), and edges E represent connections between layers. The function $\mathcal{F}$ is typically defined in terms of accuracy, loss, computational efficiency, and resource constraints. The search process can be formulated as a reinforcement learning (RL) problem, where an agent explores the architecture space and updates its policy using rewards derived from model performance. Let $\pi_\theta(a_t \mid s_t)$ be a policy parameterized by θ, which selects an action a_t (modifying the architecture) given state s_t (current architecture configuration). The expected reward function is then

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \tag{14.2}$$

where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is a trajectory sampled from policy π_θ, γ is a discount factor, and r_t is the reward (e.g., validation accuracy). Policy gradient methods such as REINFORCE update the policy via gradient ascent,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau) \right] \tag{14.3}$$

where $R(\tau)$ is the cumulative reward of a sampled trajectory. Alternatively, evolutionary algorithms model the search as a population-based optimization problem, evolving architectures through crossover and mutation operators. The fitness function in this context is given by

$$f(\alpha) = \mathcal{E}(\alpha, \mathcal{D}) \tag{14.4}$$

where $\mathcal{E}$ is an evaluation metric such as classification accuracy or mean squared error. Another approach is gradient-based NAS, where the architecture is parameterized by continuous variables α_i and optimized via gradient descent. If w denotes network weights and $L(w, \alpha)$ is the loss function, then the architecture optimization problem is

$$\min_{\alpha} L(w^*(\alpha), \alpha) \tag{14.5}$$

where $w^*(\alpha)$ are optimal weights obtained by solving

$$w^*(\alpha) = \arg\min_w L(w, \alpha). \tag{14.6}$$

The optimization follows a bilevel formulation, where the outer optimization updates α while the inner optimization solves for w. The gradient of $L(w^*(\alpha), \alpha)$ with respect to α is computed using implicit differentiation,

$$\nabla_\alpha L(w^*(\alpha), \alpha) = \nabla_\alpha L(w, \alpha) - \nabla_w L(w, \alpha)(\nabla_w^2 L(w, \alpha))^{-1} \nabla_\alpha w. \tag{14.7}$$

A continuous relaxation of discrete architectures is achieved using softmax over candidate operations o_i,

$$o_{\text{mixed}}(x) = \sum_i \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)} o_i(x). \tag{14.8}$$

The optimal architecture is obtained by discretizing α_i after training. The computational complexity of NAS is often mitigated by weight sharing across architectures, leading to one-shot NAS methods where a supernet $\mathcal{N}$ encompasses all possible subnets α, and training occurs over a shared weight space W,

$$\min_W \mathbb{E}_{\alpha \sim \mathcal{A}} L(W, \alpha). \tag{14.9}$$

After training, the best subnet is selected by evaluating candidate architectures sampled from $\mathcal{A}$. The performance of a neural architecture is influenced by hyperparameters λ, which affect layer depth, width, and activation functions. A common formulation includes multi-objective optimization,

$$\max_\alpha \mathbb{E}\left[\mathcal{F}(\alpha)\right] - \lambda \cdot C(\alpha) \tag{14.10}$$

where $C(\alpha)$ represents computational cost (e.g., FLOPs or latency). The search can be constrained using Lagrange multipliers,

$$\mathcal{L}(\alpha, \lambda) = \mathcal{F}(\alpha) - \lambda C(\alpha), \tag{14.11}$$

with the optimal trade-off achieved by solving

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0. \tag{14.12}$$

By iteratively refining $\hat{\mathcal{F}}$, the search converges to architectures with high performance and low computational cost. The theoretical underpinnings of NAS rely on neural tangent kernels (NTKs) to approximate training dynamics, where the NTK matrix Θ evolves as

$$\frac{d\Theta}{dt} = -\eta \Theta^2, \tag{14.13}$$

where η is the learning rate. A stable architecture satisfies the condition

$$\lambda_{\max}(\Theta) \ll 1. \tag{14.14}$$

Thus, NAS systematically optimizes architectures through structured exploration of the search space, reinforcement learning, evolutionary algorithms, differentiable search, and surrogate modeling, ensuring optimality under computational constraints.

14.1 Evolutionary Algorithms in Network Architecture Search

Evolutionary Algorithms (EAs) have emerged as a powerful approach for automating Network Architecture Search (NAS) in Deep Neural Networks (DNNs), leveraging principles of natural evolution to optimize network topologies. The core idea is to represent a neural network architecture as a set of hyperparameters and connection structures, encoding them into a genetic representation that undergoes evolutionary processes such as selection, mutation, and crossover to explore the vast search space efficiently. Given a population of architectures $\mathcal{P} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_N\}$, where each $\mathcal{N}_i$ represents a candidate neural network, we define the fitness function $f : \mathcal{N} \to \mathbb{R}$, which evaluates the performance of a network based on metrics such as validation accuracy, computational efficiency, or robustness to perturbations.

Each neural network architecture $\mathcal{N}_i$ can be described by a directed acyclic graph (DAG) $G_i = (V_i, E_i)$, where V_i represents the set of layers (e.g., convolutional layers, pooling layers, fully connected layers), and E_i represents the connections between them. A typical encoding scheme maps G_i to a vector representation $x_i \in \mathbb{R}^d$, where each element represents hyperparameters such as the number of filters F_l in a convolutional layer l, kernel sizes K_l, and activation functions σ_l. The architecture search problem can thus be formulated as an optimization problem:

$$\max_{x_i \in \mathcal{X}} f(x_i) \tag{14.15}$$

where $\mathcal{X}$ is the feasible set of architectures defined by design constraints (e.g., hardware constraints, FLOP limitations). The evolutionary process begins with the initialization of a population $\mathcal{P}^{(0)}$, where architectures are randomly generated or sampled from prior knowledge. At each generation t, a selection operator S chooses a subset of architectures $\mathcal{P}_s^{(t)} \subset \mathcal{P}^{(t)}$ based on their fitness scores:

$$\mathcal{P}_s^{(t)} = S(\mathcal{P}^{(t)}, f) \tag{14.16}$$

Common selection methods include tournament selection and rank-based selection. The selected architectures undergo crossover and mutation operations. Crossover combines two parent architectures $\mathcal{N}_a$ and $\mathcal{N}_b$ to produce an offspring $\mathcal{N}_c$ using a recombination function C:

$$\mathcal{N}_c = C(\mathcal{N}_a, \mathcal{N}_b) \tag{14.17}$$

For instance, if architectures are encoded as binary strings representing layer connections, one-point or uniform crossover can be applied:

$$x_c = (x_a^{(1:k)}, x_b^{(k+1:d)}) \tag{14.18}$$

where k is a randomly selected crossover point. Mutation introduces random variations in the offspring architecture to maintain diversity in the population. Given a mutation rate p_m, a mutation operator M perturbs the architecture encoding:

$$x_i' = M(x_i, p_m) \tag{14.19}$$

Common mutation operations include altering the number of filters F_l, modifying kernel sizes K_l, or randomly inserting/deleting layers. The updated population $\mathcal{P}^{(t+1)}$ for the next generation is formed by selecting the top-performing architectures:

$$\mathcal{P}^{(t+1)} = \text{Elitism}(\mathcal{P}_s^{(t)} \cup \mathcal{P}_c^{(t)}, f) \tag{14.20}$$

where $\mathcal{P}_c^{(t)}$ is the set of offspring architectures and elitism ensures that the best architectures persist across generations. The process repeats until a termination criterion is met, such as a maximum number of generations T or convergence in fitness scores:

$$\left| f(\mathcal{N}^{(t)}) - f(\mathcal{N}^{(t-1)}) \right| < \epsilon \tag{14.21}$$

where ϵ is a predefined tolerance threshold. The search efficiency of EAs can be enhanced using surrogate models $\hat{f}$ that approximate the fitness function:

$$\hat{f}(x) = \sum_{i=1}^{n} w_i K(x, x_i) \tag{14.22}$$

where $K(x, x_i)$ is a kernel function and w_i are weights learned from evaluated architectures. Bayesian optimization and neural predictors are often employed to guide the search towards promising regions of $\mathcal{X}$. To further improve convergence speed, weight inheritance strategies transfer trained weights from parent networks to offspring, reducing the need for full training cycles. Given a parent-offspring pair $(\mathcal{N}_p, \mathcal{N}_c)$, weight inheritance is expressed as:

$$W_c = \mathcal{T}(W_p, \mathcal{N}_p, \mathcal{N}_c) \tag{14.23}$$

where W_p and W_c denote the weight matrices of the parent and child networks, and $\mathcal{T}$ is a transformation function that maps weights based on structural similarities. Evolutionary NAS has demonstrated competitive performance against reinforcement learning-based and gradient-based approaches, particularly in discovering novel architectures for convolutional neural networks (CNNs) and transformers. The combination of evolutionary search with differentiable search spaces, such as in Differentiable Architecture Search (DARTS), leads to hybrid methods where architectures evolve within a continuous relaxation of the search space:

$$\mathcal{L}(\alpha) = \sum_i w_i \sigma(\alpha_i) \tag{14.24}$$

where α represents architecture parameters and $\sigma(\cdot)$ is a softmax function ensuring differentiability. By integrating EAs with differentiable optimization, recent methods achieve state-of-the-art performance while maintaining exploration capabilities.

14.2 Reinforcement Learning in Network Architecture Search

Reinforcement Learning (RL) plays a pivotal role in Network Architecture Search (NAS) for Deep Neural Networks (DNNs) by formulating the problem as a sequential decision-making process, wherein a policy optimizes the selection of architectural components to maximize a predefined performance metric, typically validation accuracy. The process is modeled as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ represents the state space corresponding to partially constructed architectures, $\mathcal{A}$ is the action space defining possible modifications to architectures, P is the transition probability governing the evolution of architectures, R denotes the reward function that quantifies model performance, and γ is the discount factor controlling long-term reward contributions. The objective is to optimize the policy $\pi(a|s;\theta)$, parameterized by θ, such that the expected cumulative reward is maximized:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_t \right]. \tag{14.25}$$

To optimize $J(\theta)$, policy gradient methods are employed, where the gradient of the expected reward with respect to policy parameters is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t;\theta) R_t \right]. \tag{14.26}$$

Using Monte Carlo estimates, the policy parameters are updated via gradient ascent:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t;\theta) R_t. \tag{14.27}$$

A common approach to implementing NAS with RL is the use of a Recurrent Neural Network (RNN)-based controller that generates candidate architectures sequentially. The hidden state h_t of the RNN encodes information about the architecture decisions made up to step t, and the action a_t at each step is sampled from a softmax distribution:

$$a_t \sim \text{Softmax}(Wh_t + b). \tag{14.28}$$

The reward signal is obtained by training and evaluating the generated architecture on a validation set, yielding an accuracy score A, which serves as the reward:

$$R = A - b, \tag{14.29}$$

where b is a baseline used in variance reduction techniques such as REINFORCE. The training of the controller follows the REINFORCE algorithm with baseline subtraction:

$$\theta \leftarrow \theta + \alpha(R - b) \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t; \theta). \tag{14.30}$$

To further stabilize learning, advantage estimation methods such as Generalized Advantage Estimation (GAE) can be employed, where the advantage function A_t is computed using the value function $V(s_t)$:

$$A_t = R_t + \gamma V(s_{t+1}) - V(s_t). \tag{14.31}$$

An alternative approach is to model the NAS problem as a Q-learning task, where the Q-value represents the expected future reward given a state-action pair:

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| s_t, a_t\right]. \tag{14.32}$$

The Q-values are updated using the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right). \tag{14.33}$$

In practical implementations, Deep Q-Networks (DQN) are utilized, where a neural network approximates the Q-function $Q(s, a; \theta)$, and updates are made using the loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right], \tag{14.34}$$

where θ^- represents target network parameters and D is an experience replay buffer. To address the instability of Q-learning, Double Q-learning is often employed, where two networks, Q_1 and Q_2, are used to decouple action selection and evaluation:

$$Q(s_t, a_t) = r_t + \gamma Q_{\min}(s_{t+1}, a_{t+1}), \tag{14.35}$$

where $Q_{\min}$ selects the minimum Q-value from two estimators to mitigate overestimation bias. In actor-critic methods such as Proximal Policy Optimization (PPO), the actor updates the policy while the critic evaluates its performance using the objective:

$$L(\theta) = \mathbb{E}\left[\min\left(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})} A_t, \text{clip}\left(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}, 1-\epsilon, 1+\epsilon\right) A_t\right)\right]. \tag{14.36}$$

This clipped objective prevents large policy updates, ensuring stability in the RL-based NAS framework. By iteratively refining the architecture using these RL techniques, NAS can effectively discover high-performing network topologies that outperform manually designed architectures.

14.3 Policy gradient methods in Network Architecture Search

Policy gradient methods provide a mathematically rigorous framework for optimizing network architectures in deep neural networks by formulating the architecture search as a reinforcement learning (RL) problem. The core idea is to define a probability distribution over possible network architectures, represented as a parametric policy $\pi_\theta(a|s)$, where s represents the state (e.g., the current design choices or hyperparameters of the network), and a represents an action (e.g., selecting a particular layer type, activation function, or connection pattern). The objective is to maximize the expected performance $J(\theta)$ of the architecture, which is commonly formulated as the expected reward obtained from training and evaluating the selected network:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \right] \tag{14.37}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ is a trajectory of states and actions sampled according to the policy $\pi_\theta(a|s)$, and $R(\tau)$ represents the cumulative reward (e.g., validation accuracy of the trained architecture). The distribution over trajectories is given by

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{14.38}$$

where $p(s_{t+1}|s_t, a_t)$ is the environment transition function, often deterministic in the context of architecture search. To optimize $J(\theta)$, the policy gradient theorem provides an unbiased estimator of the gradient using the likelihood ratio trick:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \nabla_\theta \log p_\theta(\tau) \right]. \tag{14.39}$$

Expanding the gradient term, we obtain

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{14.40}$$

Thus, the gradient estimate simplifies to

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \tag{14.41}$$

In practice, this expectation is estimated via Monte Carlo sampling over a set of architectures, leading to the update rule:

$$\theta \leftarrow \theta + \alpha \sum_{\tau \sim p_\theta(\tau)} R(\tau) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{14.42}$$

To improve convergence, a baseline $b(s_t)$ is often introduced to reduce variance:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} (R(\tau) - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \tag{14.43}$$

A common choice for $b(s_t)$ is a learned value function $V^\pi(s_t)$, leading to an advantage function $A(s_t, a_t) = R(\tau) - V^\pi(s_t)$, which forms the basis for actor-critic methods. The reward function $R(\tau)$ is typically designed to reflect the generalization ability of the architecture, often incorporating the validation accuracy of the trained model:

$$R(\tau) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f_{\mathcal{A}}(x_i; w^*)) \tag{14.44}$$

where $\mathcal{A}$ denotes the architecture defined by the trajectory τ, $f_{\mathcal{A}}$ represents the network function, and w^* are the trained weights obtained by minimizing the training loss

$$w^* = \arg\min_w \sum_{i=1}^{N} \ell(y_i, f_{\mathcal{A}}(x_i; w)). \tag{14.45}$$

To improve exploration, entropy regularization is often added to the objective:

$$J'(\theta) = J(\theta) + \lambda H(\pi_\theta) \tag{14.46}$$

where

$$H(\pi_\theta) = -\sum_{t=0}^{T} \sum_{a} \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \tag{14.47}$$

encourages diverse architecture sampling. In a real implementation, the policy π_θ is usually parameterized by a recurrent neural network (RNN) such as an LSTM, where the hidden state encodes past architecture decisions. The architecture parameters θ are updated using stochastic gradient descent with policy gradient updates, while the sampled architectures are trained using standard backpropagation. The convergence of policy gradient methods in architecture search relies on ensuring sufficient exploration and avoiding premature convergence to suboptimal architectures. Techniques such as proximal policy optimization (PPO), trust region policy optimization (TRPO), and natural gradient methods are often used to stabilize updates and improve sample efficiency. These methods modify the standard gradient update by introducing constraints such as

$$D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}||\pi_\theta) \leq \delta \tag{14.48}$$

where D_{KL} is the Kullback-Leibler divergence, ensuring that the updated policy does not deviate too drastically from the previous policy, thereby maintaining stability in the optimization process.

14.4 Neural tangent kernels (NTKs) in Network Architecture Search

Neural Tangent Kernels (NTKs) have emerged as a powerful theoretical framework for understanding the training dynamics and generalization properties of deep neural networks in the infinite-width limit. In the context of Network Architecture Search (NAS), NTKs provide a mathematically rigorous way to evaluate the expressivity and trainability of different neural network architectures without explicitly training them, thereby significantly reducing computational costs associated with traditional NAS methods. The NTK is formally defined as the Gram matrix of the Jacobian of the neural network outputs with respect to its parameters, capturing how perturbations in parameters influence the output function. Mathematically, if $f_\theta(x)$ denotes the output of a neural network with parameters θ for an input x, then the NTK is given by

$$\Theta(x, x') = \mathbb{E}_\theta \left[\nabla_\theta f_\theta(x) \cdot \nabla_\theta f_\theta(x')^T \right]. \tag{14.49}$$

As the width of the network approaches infinity, the NTK remains constant during training under gradient descent, allowing the network to be effectively modeled as a Gaussian process. This property enables rapid evaluation of the convergence behavior and generalization ability of a given architecture by analyzing the eigenvalues of the NTK matrix. The training dynamics of an infinitely wide neural network can be described using the evolution of the function $f_t(x)$ under gradient descent with a learning rate η, which follows

$$\frac{df_t(x)}{dt} = -\eta \sum_{i=1}^{n} \Theta(x, x_i)(f_t(x_i) - y_i). \tag{14.50}$$

Since the NTK remains constant in the infinite-width regime, this equation can be solved explicitly using

$$f_t(x) = f_0(x) + \Theta(x, X)\Theta(X, X)^{-1}(e^{-\eta\Theta(X,X)t}(y - f_0(X))), \tag{14.51}$$

where $X = \{x_1, \ldots, x_n\}$ denotes the training set inputs, and $y = \{y_1, \ldots, y_n\}$ are the corresponding targets. The rate of convergence is determined by the smallest eigenvalue λ_{min} of $\Theta(X, X)$, implying that architectures with larger minimum eigenvalues of their NTKs converge faster and are thus more trainable. The NTK perspective enables a principled approach to NAS by evaluating candidate architectures based on their kernel spectra, particularly the condition number $\kappa(\Theta) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$, which governs the stability of gradient descent. Generalization in deep learning is often characterized by the smoothness and complexity of learned functions. The NTK controls the function space induced by a given architecture, allowing estimation of generalization error through the kernel ridge regression formula

$$\mathbb{E}_{\text{test}}\left[(f(x_{\text{test}}) - y_{\text{test}})^2\right] \approx \sigma^2 \text{Tr}\left(\Theta(X, X)^{-1}\Theta(X_{\text{test}}, X_{\text{test}})\Theta(X, X)^{-1}\right), \tag{14.52}$$

where σ^2 is the noise variance. Architectures with better generalization properties exhibit smaller trace norms of the NTK inverse. In the infinite-width limit, NTKs of different architectures can be computed analytically by considering the recursive propagation of covariance matrices through network layers. For a fully connected network with ReLU activation, the NTK at depth L is given by

$$\Theta^{(L)}(x, x') = \Theta^{(L-1)}(x, x')\dot{K}^{(L)}(x, x'), \tag{14.53}$$

where $\dot{K}^{(L)}(x, x')$ is the derivative of the activation function's covariance kernel. Different architectures, such as convolutional neural networks (CNNs), modify this recurrence relation by incorporating weight-sharing and local connectivity constraints. The NTK for a CNN layer with filter size k is

$$\Theta^{(L)}_{\text{conv}}(x, x') = \frac{1}{k^2}\sum_{i,j}\Theta^{(L-1)}(x_i, x'_j)\dot{K}^{(L)}(x_i, x'_j). \tag{14.54}$$

This allows NAS to be performed analytically by evaluating how architectural choices influence the NTK and its spectral properties, bypassing the need for expensive training runs. Recent work has extended NTK-based NAS to attention-based architectures, such as transformers, where the self-attention mechanism induces a structured NTK

$$\Theta_{\text{attn}}(x, x') = \mathbb{E}\left[\text{softmax}\left(\frac{Q(x)K(x')^T}{\sqrt{d}}\right)V(x)V(x')^T\right], \tag{14.55}$$

where Q, K, V are the query, key, and value matrices. The spectrum of Θ_{attn} determines the expressivity and trainability of transformer-based models, enabling NTK-based architectural optimization in sequence learning tasks. The NTK framework provides an elegant, mathematically rigorous approach to NAS by linking architectural choices to trainability and generalization through spectral analysis of the kernel.

15 Training Neural Networks

Literature Review: Sorrenson (2025) [119] introduced a framework enabling exact maximum likelihood training of unrestricted neural networks. It presents new training methodologies based on probabilistic models and applies them to scientific applications. Liu and Shi (2015) [120] applied advanced neural network theory to meteorological predictions. It uses sensitivity analysis and new training techniques to mitigate sample size limitations. Das et. al. (2025) [121] integrated Finite Integral Transform (FIT) with gradient-enhanced physics-informed neural networks (g-PINN), optimizing training in engineering applications. Zhang et. al. (2025) [122] in his thesis explores neural tangent kernel (NTK) theory to model the gradient descent training process of deep networks and its implications for structural identification. Ali and Hussein (2025) [123] developed a hybrid approach combining fuzzy set theory and artificial neural networks, enhancing training robustness through heuristic optimization. Li (2025) [124] introduced a deep learning-based strategy to train neural networks for imperfect-information extensive-form games, emphasizing offline training techniques. Hu et. al. (2025) [125] explored the convergence properties of deep learning-based PDE solvers, analyzing training loss and function space properties. Chen et. al. (2025) [126] developed a Transformer-based neural network training framework for risk analysis, incorporating feature maps and game-theoretic interpretation. Sun et. al. (2025) [127] established a new benchmarking suite for optimizing neural architecture search (NAS) techniques in training spiking neural networks. Zhang et. al. (2025) [128] proposed a novel iterative training approach for neural networks, enhancing convergence guarantees in theory and practice.

15.1 Backpropagation Algorithm

Consider a neural network with L layers, where each layer l (with $l = 1, 2, \dots, L$) consists of a weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, a bias vector $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, and an activation function $\sigma^{(l)}$ which is applied element-wise. The network takes as input a vector $x^{(i)} \in \mathbb{R}^{n_0}$ for the i-th training sample, where n_0 is the number of input features, and propagates it through the layers to produce an output $\hat{y}^{(i)} \in \mathbb{R}^{n_L}$, where n_L is the number of output units. The network parameters (weights and biases) $\theta = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}\}$ are to be optimized to minimize a loss function that captures the error between the predicted output $\hat{y}^{(i)}$ and the true target $y^{(i)}$ for all training examples. For each training sample, we define the loss function $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ as the squared error:

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}\|\hat{y}^{(i)} - y^{(i)}\|_2^2, \tag{15.1}$$

where $\|\cdot\|_2$ represents the Euclidean norm. The total loss $J(\theta)$ for the entire dataset is the average of the individual losses:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}), \tag{15.2}$$

where N is the number of training samples. For squared error loss, we can write:

$$J(\theta) = \frac{1}{2N}\sum_{i=1}^{N} \|\hat{y}^{(i)} - y^{(i)}\|_2^2. \tag{15.3}$$

The forward pass through the network consists of computing the activations at each layer. For the l-th layer, the pre-activation $z^{(l)}$ is calculated as:

$$z^{(l)} = \mathbf{W}^{(l)} a^{(l-1)} + \mathbf{b}^{(l)}, \tag{15.4}$$

where $a^{(l-1)}$ is the activation from the previous layer and $\mathbf{W}^{(l)}$ is the weight matrix connecting the $(l-1)$-th layer to the l-th layer. The output of the layer, i.e., the activation $a^{(l)}$, is computed by applying the activation function $\sigma^{(l)}$ element-wise to $z^{(l)}$:

$$a^{(l)} = \sigma^{(l)}(z^{(l)}). \tag{15.5}$$

The final output of the network is given by the activation $a^{(L)}$ at the last layer, which is the predicted output $\hat{y}^{(i)}$:

$$\hat{y}^{(i)} = a^{(L)}. \tag{15.6}$$

The backpropagation algorithm computes the gradient of the loss function $J(\theta)$ with respect to each parameter (weights and biases). First, we compute the error at the output layer. Let $\delta^{(L)}$ represent the error at layer L. This is computed by taking the derivative of the loss function with respect to the activations at the output layer:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \odot \sigma^{(L)'}(z^{(L)}), \tag{15.7}$$

where $\odot$ denotes element-wise multiplication, and $\sigma^{(L)'}(z^{(L)})$ is the derivative of the activation function applied element-wise to $z^{(L)}$. For squared error loss, the derivative with respect to the activations is:

$$\frac{\partial \mathcal{L}}{\partial a^{(L)}} = \hat{y}^{(i)} - y^{(i)} \tag{15.8}$$

so the error term at the output layer is:

$$\delta^{(L)} = (\hat{y}^{(i)} - y^{(i)}) \odot \sigma^{(L)'}(z^{(L)}) \tag{15.9}$$

To propagate the error backward through the network, we compute the errors at the hidden layers. For each hidden layer $l = L-1, L-2, \ldots, 1$, the error $\delta^{(l)}$ is calculated by the chain rule:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)T} \delta^{(l+1)}\right) \odot \sigma^{(l)'}(z^{(l)}) \tag{15.10}$$

where $\mathbf{W}^{(l+1)T} \in \mathbb{R}^{n_{l+1} \times n_l}$ is the transpose of the weight matrix connecting layer l to layer $l+1$. This equation uses the fact that the error at layer l depends on the error at the next layer, modulated by the weights, and the derivative of the activation function at layer l. Once the errors $\delta^{(l)}$ are computed for all layers, we can compute the gradients of the loss function with respect to the parameters (weights and biases). The gradient of the loss with respect to the weights $\mathbf{W}^{(l)}$ is:

$$\frac{\partial J(\theta)}{\partial \mathbf{W}^{(l)}} = \frac{1}{N} \sum_{i=1}^{N} \delta^{(l)} (a^{(l-1)})^T \tag{15.11}$$

The gradient of the loss with respect to the biases $\mathbf{b}^{(l)}$ is:

$$\frac{\partial J(\theta)}{\partial \mathbf{b}^{(l)}} = \frac{1}{N} \sum_{i=1}^{N} \delta^{(l)} \tag{15.12}$$

After computing these gradients, we update the parameters using an optimization algorithm such as gradient descent. The weight update rule is:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J(\theta)}{\partial \mathbf{W}^{(l)}}, \tag{15.13}$$

and the bias update rule is:

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial J(\theta)}{\partial \mathbf{b}^{(l)}} \tag{15.14}$$

where η is the learning rate controlling the step size in the gradient descent update. This process of forward pass, backpropagation, and parameter update is repeated over multiple epochs, with each epoch consisting of a forward pass, a backward pass, and a parameter update, until the network converges to a local minimum of the loss function.

At each step of backpropagation, the chain rule is applied recursively to propagate the error backward through the network, adjusting each weight and bias to minimize the total loss. The derivative of the activation function $\sigma^{(l)\prime}(z^{(l)})$ is critical, as it dictates how the error is modulated at each layer. Depending on the choice of activation function (e.g., ReLU, sigmoid, or tanh), the derivative will take different forms, and this choice has a direct impact on the learning dynamics and convergence rate of the network. Thus, backpropagation serves as the computational backbone of neural network training. By calculating the gradients of the loss function with respect to the network parameters through efficient error propagation, backpropagation allows the network to adjust its parameters iteratively, gradually minimizing the error and improving its performance across tasks. This process is mathematically rigorous, utilizing fundamental principles of calculus and optimization, ensuring that the neural network learns effectively from its training data.

15.2 Gradient Descent Variants

The training of neural networks using gradient descent and its variants is a mathematically intensive process that aims to minimize a differentiable scalar loss function $\mathcal{L}(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the parameter vector of the neural network. The loss function is often expressed as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i), \tag{15.15}$$

where $(\boldsymbol{x}_i, y_i)$ are the input-output pairs in the training dataset of size N, and $\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)$ is the sample-specific loss. The minimization problem is solved iteratively, starting from an initial guess $\boldsymbol{\theta}^{(0)}$ and updating according to the rule

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{15.16}$$

where $\eta > 0$ is the learning rate, and $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ is the gradient of the loss with respect to $\boldsymbol{\theta}$. The gradient, computed via backpropagation, follows the chain rule and propagates through the network's layers to adjust weights and biases optimally. In a feedforward neural network with L layers, the computations proceed as follows. The input to layer l is

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{15.17}$$

where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ are the weight matrix and bias vector for the layer, respectively, and $\boldsymbol{a}^{(l-1)}$ is the activation vector from the previous layer. The output is then

$$\boldsymbol{a}^{(l)} = f^{(l)}(\boldsymbol{z}^{(l)}), \tag{15.18}$$

where $f^{(l)}$ is the activation function. Backpropagation begins with the computation of the error at the output layer,

$$\boldsymbol{\delta}^{(L)} = \frac{\partial \ell}{\partial \boldsymbol{a}^{(L)}} \odot f'^{(L)}(\boldsymbol{z}^{(L)}), \tag{15.19}$$

where $f'^{(L)}(\cdot)$ is the derivative of the activation function. For hidden layers, the error propagates recursively as

$$\boldsymbol{\delta}^{(l)} = (\boldsymbol{W}^{(l+1)})^{\top} \boldsymbol{\delta}^{(l+1)} \odot f'^{(l)}(\boldsymbol{z}^{(l)}). \tag{15.20}$$

The gradients for weight and bias updates are then computed as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)}(\boldsymbol{a}^{(l-1)})^\top \tag{15.21}$$

and

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}, \tag{15.22}$$

respectively. The dynamics of gradient descent are deeply influenced by the curvature of the loss surface, encapsulated by the Hessian matrix

$$\boldsymbol{H}(\boldsymbol{\theta}) = \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{15.23}$$

For a small step size η, the change in the loss function can be approximated as

$$\Delta \mathcal{L} \approx -\eta \|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\|^2 + \frac{\eta^2}{2}(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}))^\top \boldsymbol{H}(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{15.24}$$

This reveals that convergence is determined not only by the gradient magnitude but also by the curvature of the loss surface along the gradient direction. The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ of $\boldsymbol{H}(\boldsymbol{\theta})$ dictate the local geometry, with large condition numbers $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ slowing convergence due to ill-conditioning. Stochastic gradient descent (SGD) modifies the standard gradient descent by computing updates based on a single data sample $(\boldsymbol{x}_i, y_i)$, leading to

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i). \tag{15.25}$$

While SGD introduces variance into the updates, this stochasticity helps escape saddle points characterized by zero gradient but mixed curvature. To balance computational efficiency and stability, mini-batch SGD computes gradients over a randomly selected subset $\mathcal{B} \subset \{1, \ldots, N\}$ of size $|\mathcal{B}|$, yielding

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i). \tag{15.26}$$

Momentum methods enhance convergence by incorporating a memory of past gradients. The velocity term

$$\boldsymbol{v}^{(k+1)} = \gamma \boldsymbol{v}^{(k)} + \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{15.27}$$

accumulates gradient information, and the parameter update is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \boldsymbol{v}^{(k+1)}. \tag{15.28}$$

Analyzing momentum in the eigenspace of $\boldsymbol{H}(\boldsymbol{\theta})$, with $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$, reveals that the effective step size in each eigendirection is

$$\eta_{\text{eff},i} = \frac{\eta}{1 - \gamma \lambda_i}, \tag{15.29}$$

showing that momentum accelerates convergence in low-curvature directions while damping oscillations in high-curvature directions. Adaptive gradient methods, such as AdaGrad, RMSProp, and Adam, refine learning rates for individual parameters. In AdaGrad, the adaptive learning rate is

$$\eta_i^{(k+1)} = \frac{\eta}{\sqrt{\boldsymbol{G}_{ii}^{(k+1)} + \epsilon}}, \tag{15.30}$$

where

$$\boldsymbol{G}_{ii}^{(k+1)} = \boldsymbol{G}_{ii}^{(k)} + \left(\nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{15.31}$$

RMSProp modifies this with an exponentially weighted average

$$\boldsymbol{G}_{ii}^{(k+1)} = \beta \boldsymbol{G}_{ii}^{(k)} + (1 - \beta) \left(\nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{15.32}$$

Adam combines RMSProp with momentum, where the first and second moments are

$$\boldsymbol{m}^{(k+1)} = \beta_1 \boldsymbol{m}^{(k)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \tag{15.33}$$

and

$$\boldsymbol{v}^{(k+1)} = \beta_2 \boldsymbol{v}^{(k)} + (1-\beta_2)\left(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{15.34}$$

Bias corrections yield

$$\hat{\boldsymbol{m}}^{(k+1)} = \frac{\boldsymbol{m}^{(k+1)}}{1-\beta_1^k}, \quad \hat{\boldsymbol{v}}^{(k+1)} = \frac{\boldsymbol{v}^{(k+1)}}{1-\beta_2^k}. \tag{15.35}$$

The final parameter update is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \frac{\hat{\boldsymbol{m}}^{(k+1)}}{\sqrt{\hat{\boldsymbol{v}}^{(k+1)}} + \epsilon}. \tag{15.36}$$

In conclusion, gradient descent and its variants provide a rich framework for optimizing neural network parameters. While standard gradient descent offers a basic approach, advanced methods like momentum and adaptive gradients significantly enhance convergence by tailoring updates to the landscape of the loss surface and the dynamics of training.

15.2.1 SGD (Stochastic Gradient Descent) Optimizer

Literature Review: Lauand and Meyn (2025) [175] established a theoretical framework for SGD using Markovian dynamics to improve convergence properties. It integrates quasi-periodic linear systems into SGD, enhancing its robustness in non-stationary environments. Maranjyan et al. (2025) [176] developed an asynchronous SGD algorithm that meets the theoretical lower bounds for time complexity. It introduces ring-based communication to optimize parallel execution without degrading convergence rates. Gao and Gündüz (2025) [177] proposed a stochastic gradient descent-based approach to optimize graph neural networks in wireless networks. It rigorously analyzes the stochastic optimization problem and proves its convergence guarantees. Yoon et. al. (2025) [178] investigated federated SGD in multi-agent learning and derives theoretical guarantees on its communication efficiency while achieving equilibrium. Verma and Maiti (2025) [179] proposed a periodic learning rate (using sine and cosine functions) for SGD-based optimizers, theoretically proving its benefits in stability and computational efficiency. Borowski and Miasojedow (2025) [180] extended the Robbins-Monro theorem to analyze convergence guarantees of SGD, refining the theoretical understanding of projected stochastic approximation algorithms. Dong et al. (2025) [181] applied stochastic gradient descent to brain network modeling, providing a theoretical framework for optimizing neural control strategies. Jiang et. al. (2025) [182] analyzed the bias-variance tradeoff in decentralized SGD, proving convergence rates and proposing an error-correction mechanism for biased gradients. Sonobe et. al. (2025) [183] connected SGD with Bayesian inference, presenting a theoretical analysis of how stochastic optimization methods approximate posterior distributions. Zhang and Jia (2025) [184] examined the theoretical properties of policy gradients in reinforcement learning, proving convergence guarantees for stochastic optimal control problems.

The **Stochastic Gradient Descent (SGD) optimizer** is an iterative method designed to minimize an objective function $f(\mathbf{w})$ by updating a parameter vector $\mathbf{w}$ in the direction of the negative gradient. The fundamental optimization problem can be expressed as

$$\min_{\mathbf{w}} f(\mathbf{w}), \tag{15.37}$$

where

$$f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; \mathbf{x}_i, y_i) \tag{15.38}$$

represents the empirical risk, constructed from a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Here, $\ell(\mathbf{w}; \mathbf{x}_i, y_i)$ denotes the loss function, $\mathbf{w} \in \mathbb{R}^d$ is the parameter vector, N is the dataset size, and $f(\mathbf{w})$ approximates the true population risk

$$\mathbb{E}_{\mathbf{x},y}[\ell(\mathbf{w}; \mathbf{x}, y)]. \tag{15.39}$$

Standard gradient descent involves the update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)}), \tag{15.40}$$

where $\eta > 0$ is the learning rate and

$$\nabla f(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(\mathbf{w}; \mathbf{x}_i, y_i) \tag{15.41}$$

is the full gradient. However, for large-scale datasets, the computation of $\nabla f(\mathbf{w})$ becomes computationally prohibitive, motivating the adoption of stochastic approximations. The stochastic approximation relies on the idea of estimating the gradient $\nabla f(\mathbf{w})$ using a single data point or a small batch of data points. Denoting the random index sampled at iteration t as i_t, the stochastic gradient can be written as

$$\widehat{\nabla} f(\mathbf{w}^{(t)}) = \nabla \ell(\mathbf{w}^{(t)}; \mathbf{x}_{i_t}, y_{i_t}). \tag{15.42}$$

Consequently, the update rule becomes

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \widehat{\nabla} f(\mathbf{w}^{(t)}). \tag{15.43}$$

For a mini-batch $\mathcal{B}_t$ of size m, the stochastic gradient generalizes to

$$\widehat{\nabla} f(\mathbf{w}^{(t)}) = \frac{1}{m} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}^{(t)}; \mathbf{x}_i, y_i). \tag{15.44}$$

An important property of $\widehat{\nabla} f(\mathbf{w})$ is its unbiasedness:

$$\mathbb{E}[\widehat{\nabla} f(\mathbf{w})] = \nabla f(\mathbf{w}). \tag{15.45}$$

However, the variance of $\widehat{\nabla} f(\mathbf{w})$, defined as

$$\mathrm{Var}[\widehat{\nabla} f(\mathbf{w})] = \mathbb{E}[\|\widehat{\nabla} f(\mathbf{w}) - \nabla f(\mathbf{w})\|^2], \tag{15.46}$$

introduces stochastic noise into the updates, where $\mathrm{Var}[\widehat{\nabla} f(\mathbf{w})] \approx \frac{\sigma^2}{m}$ and

$$\sigma^2 = \mathbb{E}[\|\nabla \ell(\mathbf{w}; \mathbf{x}) - \nabla f(\mathbf{w})\|^2] \tag{15.47}$$

is the variance of the gradients. To analyze the convergence properties of SGD, we assume $f(\mathbf{w})$ to be L-smooth, meaning

$$\|\nabla f(\mathbf{w}_1) - \nabla f(\mathbf{w}_2)\| \leq L \|\mathbf{w}_1 - \mathbf{w}_2\|, \tag{15.48}$$

and $f(\mathbf{w})$ to be bounded below by $f^* = \inf_{\mathbf{w}} f(\mathbf{w})$. Using Taylor expansion, we can write

$$f(\mathbf{w}^{(t+1)}) \leq f(\mathbf{w}^{(t)}) - \eta \|\nabla f(\mathbf{w}^{(t)})\|^2 + \frac{\eta^2 L}{2} \|\widehat{\nabla} f(\mathbf{w}^{(t)})\|^2. \tag{15.49}$$

Taking expectations yields

$$\mathbb{E}[f(\mathbf{w}^{(t+1)})] \leq \mathbb{E}[f(\mathbf{w}^{(t)})] - \frac{\eta}{2} \mathbb{E}[\|\nabla f(\mathbf{w}^{(t)})\|^2] + \frac{\eta^2 L}{2} \sigma^2, \tag{15.50}$$

showing that the convergence rate depends on the interplay between the learning rate η, the smoothness constant L, and the gradient variance σ^2. For η small enough, the dominant term in convergence is $-\frac{\eta}{2}\mathbb{E}[\|\nabla f(\mathbf{w}^{(t)})\|^2]$, leading to monotonic decrease in $f(\mathbf{w}^{(t)})$. In the strongly convex case, where $f(\mathbf{w})$ satisfies

$$f(\mathbf{w}_1) \geq f(\mathbf{w}_2) + \nabla f(\mathbf{w}_2)^\top(\mathbf{w}_1 - \mathbf{w}_2) + \frac{\mu}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \tag{15.51}$$

for $\mu > 0$, SGD achieves linear convergence. Specifically,

$$\mathbb{E}[\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2] \leq (1 - \eta\mu)^t\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\eta\sigma^2}{2\mu}. \tag{15.52}$$

For non-convex functions, where $\nabla^2 f(\mathbf{w})$ can have both positive and negative eigenvalues, SGD may converge to a local minimizer or saddle point. Stochasticity plays a pivotal role in escaping strict saddle points $\mathbf{w}_s$ where $\nabla f(\mathbf{w}_s) = 0$ but $\lambda_{\min}(\nabla^2 f(\mathbf{w}_s)) < 0$.

15.2.2 Nesterov Accelerated Gradient Descent (NAG)

Literature Review: The field of Nesterov Accelerated Gradient Descent (NAG) has undergone significant theoretical refinement and practical adaptation in recent years, with researchers delving into its convergence properties, dynamical systems interpretations, stochastic extensions, and domain-specific optimizations. Adly and Attouch (2024) [422] provide an in-depth complexity analysis by precisely tuning the viscosity parameter within an inertial gradient system, thereby extending NAG's classical formulations into the Su-Boyd-Candès dynamical framework. By embedding NAG within an inertial differential equation paradigm, they rigorously establish how varying the viscosity parameter alters convergence rates and acceleration effects, bridging a crucial gap between continuous-time inertial flow models and discrete-time iterative schemes. Expanding on this inertial dynamics perspective, Wang and Peypouquet (2024) [423] focus specifically on strongly convex functions, where they derive an exact convergence rate for NAG by constructing a novel Lyapunov function. Unlike previous results that provided only upper-bound estimates for convergence, their approach offers a precise characterization of NAG's asymptotic behavior, reinforcing its accelerated rate of $O(\frac{1}{k^2})$ in smooth, strongly convex settings. Their work strengthens the geometric interpretation of NAG as a discretization of a second-order differential equation with damping, further cementing its connection to continuous-time optimization dynamics.

Despite the theoretical consensus on NAG's superiority in convex optimization, Hermant et. al. (2024) [424] present an unexpected empirical and theoretical challenge to this assumption. Their study systematically compares deterministic NAG with Stochastic Gradient Descent (SGD) under convex function interpolation, revealing cases where SGD exhibits superior practical performance despite lacking formal acceleration guarantees. Their findings raise fundamental questions about the practical advantages of momentum-based methods in data-driven scenarios, particularly when stochastic noise interacts with interpolation dynamics. Applying NAG beyond classical convex optimization, Alavala and Gorthi (2024) [425] integrate it into medical imaging reconstruction, specifically for Cone Beam Computed Tomography (CBCT). They develop a NAG-accelerated least squares solver (NAG-LS), demonstrating substantial improvements in computational efficiency and image reconstruction quality. Their results indicate that NAG's ability to mitigate error propagation in iterative reconstruction algorithms makes it particularly well-suited for inverse problems in medical imaging. From a generalization perspective, Li (2024) [426] formulates a unified momentum framework encompassing NAG, Polyak's Heavy Ball method, and other stochastic momentum algorithms. By introducing a generalized momentum differential equation, he rigorously dissects the trade-off between stability, acceleration, and variance control in momentum-based optimization. His framework provides a cohesive theoretical structure for understanding how momentum-based techniques interact with gradient noise, particularly in high-dimensional stochastic settings.

Beyond convexity, Gupta and Wojtowytsch (2024) [427] rigorously analyze NAG's performance in non-convex optimization landscapes, a setting where standard acceleration techniques are often assumed ineffective. Their research establishes conditions under which NAG retains acceleration benefits even in the absence of strong convexity, highlighting how NAG's momentum interacts with saddle points, sharp local minima, and benign non-convex structures. Their work provides a crucial extension of NAG beyond convex functions, opening new avenues for its application in deep learning and high-dimensional optimization. Meanwhile, Razzouki et. al. (2024) [428] compile a comprehensive survey of gradient-based optimization methods, systematically comparing NAG, Adam, RMSprop, and other modern optimizers. Their analysis delves into theoretical convergence guarantees, empirical performance benchmarks, and practical tuning considerations, emphasizing how NAG's momentum-driven updates compare against adaptive learning rate strategies. Their survey serves as an authoritative reference for researchers seeking to navigate the landscape of momentum-based optimization algorithms. Shifting towards hardware implementations, Wang et al. (2025) [429] apply NAG to digital background calibration in Analog-to-Digital Converters (ADCs). Their study demonstrates how NAG accelerates error correction algorithms in high-speed ADC architectures, particularly in mitigating nonlinear distortions and improving signal-to-noise ratios (SNRs). Their results provide compelling evidence that momentum-based optimization transcends software applications, finding practical utility in high-performance electronic circuit design.

To further explore empirical performance trade-offs, Naeem et. al. (2024) [430] conduct an exhaustive empirical evaluation of NAG, Adam, and Gradient Descent across various convex and non-convex loss functions. Their results highlight that while NAG accelerates convergence in many cases, it can induce oscillatory behavior in certain settings, necessitating adaptive momentum tuning to prevent divergence. Their findings offer practical insights into optimizer selection strategies, particularly in deep learning architectures where gradient curvature varies dynamically. Finally, Campos et. al. (2024) [431] extend NAG to optimization on Lie groups, a fundamental class of non-Euclidean geometries. By adapting momentum-based gradient descent methods to Lie algebra structures, they establish new convergence guarantees for optimization problems on curved manifolds, an area crucial to robotics, physics, and differential geometry applications. Their work signifies a major extension of NAG's applicability, proving its efficacy beyond Euclidean space.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function with a unique minimizer:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} f(\theta). \tag{15.53}$$

We assume the L-Lipschitz Continuity of the Gradient

$$\|\nabla f(\theta) - \nabla f(\theta')\| \leq L\|\theta - \theta'\|, \quad \forall \theta, \theta' \in \mathbb{R}^d. \tag{15.54}$$

and the Strong Convexity of $f(\theta)$ with Parameter m

$$f(\theta') \geq f(\theta) + \nabla f(\theta)^T(\theta' - \theta) + \frac{m}{2}\|\theta' - \theta\|^2, \quad \forall \theta, \theta' \in \mathbb{R}^d. \tag{15.55}$$

The strong convexity assumption ensures that the Hessian satisfies:

$$mI \preceq \nabla^2 f(\theta) \preceq LI, \quad \forall \theta \in \mathbb{R}^d. \tag{15.56}$$

In Gradient Descent, Classical gradient descent updates follow:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t). \tag{15.57}$$

This method achieves a linear convergence rate of $O(1/t)$ in the convex case. In Momentum-Based Gradient Descent, the momentum-based update rule is:

$$v_t = \mu v_{t-1} - \eta \nabla f(\theta_t), \tag{15.58}$$

$$\theta_{t+1} = \theta_t + v_t. \tag{15.59}$$

where v_t is a velocity-like term accumulating past gradients. μ is the momentum coefficient. Momentum reduces oscillations and accelerates convergence but suffers from excessive oscillations in ill-conditioned problems. The Nesterov Accelerated Gradient (NAG) is A Look-Ahead Strategy. Instead of computing the gradient at θ_t, NAG applies momentum first:

$$\tilde{\theta}_t = \theta_t + \mu v_{t-1}. \tag{15.60}$$

Then, the velocity update is performed using the gradient at $\tilde{\theta}_t$:

$$v_t = \mu v_{t-1} - \eta \nabla f(\tilde{\theta}_t). \tag{15.61}$$

Finally, the parameter update follows:

$$\theta_{t+1} = \theta_t + v_t. \tag{15.62}$$

The Interpretation of the Nesterov Accelerated Gradient (NAG) is

- **Look-Ahead Gradient Computation:** By computing $\nabla f(\tilde{\theta}_t)$ instead of $\nabla f(\theta_t)$, NAG effectively anticipates the next move, leading to improved convergence rates.
- **Adaptive Step Size:** The effective step size is modified dynamically, stabilizing the trajectory.

To find the Variational Formulation of NAG, We derive NAG from an auxiliary optimization problem that minimizes an upper bound on $f(\theta)$. Define a quadratic approximation at the look-ahead iterate $\tilde{\theta}_t$:

$$\theta_{t+1} = \arg\min_{\theta} \left[f(\tilde{\theta}_t) + \nabla f(\tilde{\theta}_t)^T (\theta - \tilde{\theta}_t) + \frac{1}{2\eta} \|\theta - \theta_t\|^2 \right]. \tag{15.63}$$

Solving for θ_{t+1}:

$$\theta_{t+1} = \tilde{\theta}_t - \eta \nabla f(\tilde{\theta}_t). \tag{15.64}$$

This derivation justifies why NAG achieves adaptive step-size behavior. We analyze the convergence properties and Optimality Rate under convexity assumptions of Gradient Descent (GD). For gradient descent:

$$f(\theta_t) - f(\theta^*) = O\left(\frac{1}{t}\right). \tag{15.65}$$

This is suboptimal in large-scale settings. Regarding the NAG Convergence Rate, for strongly convex $f(\theta)$:

$$f(\theta_t) - f(\theta^*) = O\left(\frac{1}{t^2}\right). \tag{15.66}$$

This improvement is due to the momentum-enhanced look-ahead updates. We need to do the Lyapunov Analysis for Stability. Define the Lyapunov function:

$$V_t = f(\theta_t) - f(\theta^*) + \frac{\gamma}{2} \|\theta_t - \theta^*\|^2. \tag{15.67}$$

Here, $\gamma, \delta > 0$ are parameters chosen to ensure V_t is non-increasing. We analyze $V_{t+1} - V_t$ to show it is non-positive. Expanding V_{t+1}:

$$V_{t+1} = f(\theta_{t+1}) - f(\theta^*) + \frac{\gamma}{2} \|\theta_{t+1} - \theta^*\|^2 + \frac{\delta}{2} \|v_{t+1}\|^2. \tag{15.68}$$

Using strong convexity:

$$f(\theta_{t+1}) \leq f(\theta_t) + \nabla f(\theta_t)^T (\theta_{t+1} - \theta_t) + \frac{L}{2} \|\theta_{t+1} - \theta_t\|^2. \tag{15.69}$$

Since $\theta_{t+1} = \theta_t + v_t$, we substitute:

$$f(\theta_{t+1}) \leq f(\theta_t) + \nabla f(\theta_t)^T v_t + \frac{L}{2}\|v_t\|^2. \tag{15.70}$$

Now, using $v_t = \mu v_{t-1} - \eta \nabla f(\tilde{\theta}_t)$, we analyze the term $\|\theta_{t+1} - \theta^*\|^2$:

$$\|\theta_{t+1} - \theta^*\|^2 = \|\theta_t - \theta^* + v_t\|^2. \tag{15.71}$$

Expanding:

$$\|\theta_{t+1} - \theta^*\|^2 = \|\theta_t - \theta^*\|^2 + 2(\theta_t - \theta^*)^T v_t + \|v_t\|^2. \tag{15.72}$$

Similarly, we expand $\|v_{t+1}\|^2$:

$$\|v_{t+1}\|^2 = \|\mu v_t - \eta \nabla f(\tilde{\theta}_{t+1})\|^2. \tag{15.73}$$

Expanding:

$$\|v_{t+1}\|^2 = \mu^2\|v_t\|^2 - 2\mu\eta v_t^T \nabla f(\tilde{\theta}_{t+1}) + \eta^2\|\nabla f(\tilde{\theta}_{t+1})\|^2. \tag{15.74}$$

We have to choose γ, δ to Ensure Descent. To ensure $V_{t+1} \leq V_t$, we require:

$$V_{t+1} - V_t \leq 0. \tag{15.75}$$

After substituting the above expansions and simplifying, we obtain a sufficient condition:

$$\gamma \geq \frac{L}{\eta}, \quad \delta \geq \frac{1}{\eta}. \tag{15.76}$$

Choosing γ, δ appropriately, we conclude:

$$V_{t+1} \leq V_t \tag{15.77}$$

which proves the global stability of NAG. In conclusion, since V_t is non-increasing and lower-bounded (by 0), it converges, which implies that $\theta_t \to \theta^*$ and the NAG iterates remain bounded. Hence, we have rigorously proven the global stability of Nesterov's Accelerated Gradient (NAG). For Practical Considerations, we need to have:

- **Choice of μ:** Optimal momentum is $\mu = 1 - O(1/t)$.
- **Adaptive Learning Rate:** Choosing $\eta = O(1/L)$ ensures convergence.

15.2.3 Adam (Adaptive Moment Estimation) Optimizer

Literature Review: Kingma and Ba (2014) [165] introduced the Adam optimizer. It presents Adam as an adaptive gradient-based optimization method that combines momentum and adaptive learning rate techniques. The authors rigorously prove its advantages over traditional optimizers such as SGD and RMSProp. Reddy et. al. (2019) [166] analyzed the convergence properties of Adam and identified cases where it may fail to converge. The authors propose AMSGrad, an improved variant of Adam that guarantees better theoretical convergence behavior. Jin et. al. (2024) [167] introduced MIAdam (Multiple Integral Adam), which modified Adam's update rules to enhance generalization. The authors theoretically and empirically demonstrate its effectiveness in avoiding sharp minima. Adly et. al. (2024) [168] proposed EXAdam, an improvement over Adam that uses cross-moments in parameter updates. This leads to faster convergence while maintaining the adaptability of Adam. Theoretical derivations show improved variance reduction in updates. Liu et. al. (2024) [169] provided a rigorous mathematical proof of convergence for Adam when applied to linear inverse problems. The authors compare Adam's convergence rate with standard gradient descent and prove its efficiency in noisy settings. Yang (2025) [170] generalized Adam by introducing a biased stochastic optimization framework. The authors show that under specific conditions, Adam's bias correction step is insufficient, leading to poor convergence on strongly convex functions. Park and Lee (2024) [171] developed SMMF, a novel variant of Adam that factorizes momentum tensors, reducing memory usage. Theoretical bounds show that SMMF preserves Adam's adaptability while improving efficiency. Mahjoubi et al. (2025) [172] provided a comparative analysis of Adam, SGD, and RMSProp in deep learning models. It demonstrates scenarios where Adam outperforms other methods, particularly in high-dimensional optimization problems. Seini and Adam (2024) [173] examined how Adam's optimization framework can be adapted to human-AI collaborative learning models. The paper provides a theoretical foundation for integrating Adam into AI-driven education platforms. Teessar (2024) [174] discussed Adam's application in survey and social science research, where adaptive optimization is used to fine-tune questionnaire analysis models. This highlights Adam's versatility outside deep learning.

The Adaptive Moment Estimation (Adam) optimizer can be considered a sophisticated, hybrid optimization algorithm combining elements of momentum-based methods and adaptive learning rate techniques, which is why it has become a cornerstone in the optimization of complex machine learning models, particularly those used in deep learning. Adam's formulation is centered on computing and using both the first and second moments (i.e., the mean and the variance) of the gradient with respect to the loss function at each parameter update. This process effectively adapts the learning rate for each parameter, based on its respective gradient's statistical properties. The moment-based adjustments provide robustness against issues such as poor conditioning of the objective function and gradient noise, which are prevalent in large-scale optimization problems.

We aim to minimize a stochastic objective function $f(\theta)$, where $\theta \in \mathbb{R}^d$ represents the parameters of the model. The optimization problem is:

$$\theta^* = \arg\min_{\theta} \mathbb{E}[f(\theta;\xi)] \tag{15.78}$$

where ξ is a random variable representing the stochasticity (e.g., mini-batch sampling in deep learning). The Adam optimizer maintains that the first moment estimate (exponentially decaying average of gradients) is given by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{15.79}$$

where $g_t = \nabla_\theta f(\theta_{t-1};\xi_t)$ is the stochastic gradient at time t, and $\beta_1 \in [0,1)$ is the decay rate. The second moment estimate (exponentially decaying average of squared gradients) is given by:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{15.80}$$

where $\beta_2 \in [0,1)$ is the decay rate, and g_t^2 denotes element-wise squaring. The bias-corrected estimates are:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{15.81}$$

The parameter update rule is:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \tag{15.82}$$

where η is the learning rate, and $\epsilon > 0$ is a small constant for numerical stability. To rigorously analyze Adam, we impose the following assumptions. The gradient $\nabla_\theta f(\theta)$ is Lipschitz continuous with constant L:

$$\|\nabla_\theta f(\theta_1) - \nabla_\theta f(\theta_2)\| \le L\|\theta_1 - \theta_2\| \tag{15.83}$$

The stochastic gradients g_t are bounded almost surely:

$$\|g_t\|_\infty \le G \tag{15.84}$$

The second moments of the gradients are bounded:

$$\mathbb{E}[\|g_t\|^2] \leq \sigma^2 \tag{15.85}$$

The feasible region Θ is bounded with diameter D:

$$\|\theta_1 - \theta_2\| \leq D, \quad \forall \theta_1, \theta_2 \in \Theta \tag{15.86}$$

The decay rates β_1 and β_2 satisfy $0 \leq \beta_1, \beta_2 < 1$, and $\beta_1 < \beta_2$. We analyze Adam in the online optimization framework, where the loss function $f_t(\theta)$ is revealed sequentially. The goal is to bound the regret:

$$R(T) = \sum_{t=1}^{T} f_t(\theta_t) - \min_{\theta} \sum_{t=1}^{T} f_t(\theta) \tag{15.87}$$

The regret can be decomposed as:

$$R(T) = \underbrace{\sum_{t=1}^{T} f_t(\theta_t) - \sum_{t=1}^{T} f_t(\theta^*)}_{\text{Regret due to optimization}} + \underbrace{\sum_{t=1}^{T} f_t(\theta^*) - \min_{\theta} \sum_{t=1}^{T} f_t(\theta)}_{\text{Regret due to stochasticity}} \tag{15.88}$$

Regarding the Boundedness of $\hat{m}_t$ and $\hat{v}_t$, using the boundedness of g_t, we can show:

$$\|\hat{m}_t\|_\infty \leq \frac{G}{1-\beta_1}, \quad \|\hat{v}_t\|_\infty \leq \frac{G^2}{1-\beta_2} \tag{15.89}$$

The bias-corrected estimates satisfy:

$$\mathbb{E}[\hat{m}_t] = \mathbb{E}[g_t], \quad \mathbb{E}[\hat{v}_t] = \mathbb{E}[g_t^2] \tag{15.90}$$

The update rule scales the gradient by $\frac{1}{\sqrt{\hat{v}_t + \epsilon}}$, which adapts to the curvature of the loss function. Under the assumptions, the regret of Adam can be bounded as:

$$R(T) \leq \frac{D^2 T}{2\eta(1-\beta_1)} + \frac{\eta(1+\beta_1)G^2}{(1-\beta_1)(1-\beta_2)(1-\gamma)^2} \tag{15.91}$$

where $\gamma = \frac{\beta_1}{\beta_2}$. This bound is $O(\sqrt{T})$, which is optimal for online convex optimization. Regarding Convergence in Non-Convex Settings, for non-convex optimization, we analyze the convergence of Adam to a stationary point. Specifically, we show that:

$$\lim_{T \to \infty} \mathbb{E}[\|\nabla f(\theta_T)\|^2] = 0 \tag{15.92}$$

Define the Lyapunov function:

$$V_t = f(\theta_t) + \frac{\eta}{2}\|\hat{m}_t\|^2 \tag{15.93}$$

Using the Lipschitz continuity of $\nabla f(\theta)$ and the boundedness of $\hat{m}_t$ and $\hat{v}_t$, we derive:

$$\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\theta_t)\|^2\right] \leq C, \tag{15.94}$$

where C is a constant depending on $\eta, \beta_1, \beta_2, G$, and σ. As $T \to \infty$, the expected gradient norm converges to zero:

$$\mathbb{E}\left[\|\nabla f(\theta_T)\|^2\right] \to 0. \tag{15.95}$$

In conclusion, the Adam optimizer is a rigorously analyzed algorithm with strong theoretical guarantees. Its adaptive learning rates and momentum-like behavior make it highly effective for both convex and non-convex optimization problems.

Mathematically, at each iteration t, the Adam optimizer updates the parameter vector $\theta_t \in \mathbb{R}^n$, where n is the number of parameters of the model, based on the gradient g_t, which is the gradient of the objective function with respect to θ_t, i.e., $g_t = \nabla_\theta f(\theta_t)$. In its essence, Adam computes two distinct quantities: the first moment estimate m_t and the second moment estimate v_t, which are recursive moving averages of the gradients and the squared gradients, respectively. The first moment estimate m_t is given by

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{15.96}$$

where $\beta_1 \in [0, 1)$ is the decay rate for the first moment. This recurrence equation represents a weighted moving average of the gradients, which is intended to capture the *directional momentum* in the optimization process. By incorporating the first moment, Adam accumulates information about the historical gradients, which helps mitigate oscillations and stabilizes the convergence direction. The term $(1 - \beta_1)$ ensures that the most recent gradient g_t receives a more significant weight in the computation of m_t. Similarly, the second moment estimate v_t, which represents the exponentially decaying average of the squared gradients, is updated as

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{15.97}$$

where $\beta_2 \in [0, 1)$ is the decay rate for the second moment. This moving average of squared gradients captures the *variance* of the gradient at each iteration. The second moment v_t thus acts as an estimate of the *curvature* of the objective function, which allows the optimizer to adjust the step size for each parameter accordingly. Specifically, large values of v_t correspond to parameters that experience high gradient variance, signaling a need for smaller updates to prevent overshooting, while smaller values of v_t correspond to parameters with low gradient variance, where larger updates are appropriate. This mechanism is akin to automatically tuning the learning rate for each parameter based on the local geometry of the loss function. At initialization, both m_t and v_t are typically set to zero. This initialization introduces a bias toward zero, particularly at the initial time steps, causing the estimates of the moments to be somewhat underrepresented in the early iterations. To correct for this bias, *bias correction* terms are introduced. The bias-corrected first moment $\hat{m}_t$ is given by

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{15.98}$$

and the bias-corrected second moment $\hat{v}_t$ is given by

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{15.99}$$

The purpose of these corrections is to offset the initial tendency of m_t and v_t to underestimate the true values due to their initialization at zero. As the iteration progresses, the bias correction terms become less significant, and the estimates of the moments converge to their true values, allowing for more accurate parameter updates. The actual update rule for the parameters θ_t is determined by using the bias-corrected first and second moment estimates $\hat{m}_t$ and $\hat{v}_t$, respectively. The update equation is given by

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{15.100}$$

where η is the global learning rate, and ϵ is a small constant (typically 10^{-8}) added to the denominator for numerical stability. This update rule incorporates both the momentum (through $\hat{m}_t$) and the adaptive learning rate (through $\hat{v}_t$). The factor $\sqrt{\hat{v}_t} + \epsilon$ is particularly crucial as it ensures that parameters with large gradient variance (i.e., those with large values in v_t) receive smaller updates, whereas parameters with smaller gradient variance (i.e., those with small values in v_t) receive larger updates, thus preventing divergence in high-variance regions.

The learning rate adjustment in Adam is dynamic in nature, as it is controlled by the second moment estimate $\hat{v}_t$, which means that Adam has a per-parameter learning rate for each parameter. For each parameter, the learning rate is inversely proportional to the square root of its corresponding second moment estimate $\hat{v}_t$, leading to *adaptive learning rates*. This is what enables Adam to operate effectively in highly non-convex optimization landscapes, as it reduces the learning rate in directions where the gradient exhibits high variance, thus stabilizing the updates, and increases the learning rate where the gradient variance is low, speeding up convergence. In the case where Adam is applied to convex objective functions, convergence can be analyzed mathematically. Under standard assumptions, such as bounded gradients and a decreasing learning rate, the convergence of Adam can be shown by proving that

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \eta_t = \infty, \tag{15.101}$$

where η_t is the learning rate at time step t. The first condition ensures that the learning rate decays sufficiently rapidly to guarantee convergence, while the second ensures that the learning rate does not decay too quickly, allowing for continual updates as the algorithm progresses. However, Adam is not without its limitations. One notable issue arises from the fact that the second moment estimate v_t may decay too quickly, causing overly aggressive updates in regions where the gradient variance is relatively low. To address this, the AMSGrad variant was introduced. AMSGrad modifies the second moment update rule by replacing v_t with

$$\hat{v}_t^{\text{new}} = \max(\hat{v}_{t-1}, \hat{v}_t), \tag{15.102}$$

thereby ensuring that $\hat{v}_t$ never decreases, which helps prevent the optimizer from making overly large updates in situations where the second moment estimate may be miscalculated. By forcing $\hat{v}_t$ to increase or remain constant, AMSGrad reduces the chance of large, destabilizing parameter updates, thereby improving the stability and convergence of the optimizer, particularly in difficult or ill-conditioned optimization problems. Additionally, further extensions of Adam, such as AdaBelief, introduce additional modifications to the second moment estimate by introducing a belief-based mechanism to correct the moment estimates. Specifically, AdaBelief estimates the second moment $\hat{v}_t$ in a way that adjusts based on the *belief* in the direction of the gradient, offering further stability in cases where gradients may be sparse or noisy. These innovations underscore the flexibility of Adam and its variants in optimizing complex loss functions across a range of machine learning tasks.

Ultimately, the Adam optimizer stands as a highly sophisticated, mathematically rigorous optimization algorithm, effectively combining momentum and adaptive learning rates. By using both the first and second moments of the gradient, Adam dynamically adjusts the parameter updates, providing a robust and efficient optimization framework for non-convex, high-dimensional objective functions. The use of bias correction, coupled with the adaptive nature of the optimizer, allows it to operate effectively across a wide range of problem settings, making it a go-to method for many machine learning and deep learning applications. The mathematical rigor behind Adam ensures that it remains a highly stable and efficient optimization technique, capable of overcoming many of the challenges posed by large-scale and noisy gradient information in machine learning models.

15.2.4 RMSProp (Root Mean Squared Propagation) Optimizer

Literature Review: Bensaid et. al. (2024) [155] provides a rigorous analysis of the convergence properties of RMSProp under non-convex settings. It utilizes stability theory to examine how RMSProp adapts to different loss landscapes and demonstrates how adaptivity plays a crucial role in ensuring convergence. The study offers theoretical insights into the efficiency of RMSProp in smoothing out noisy gradients. Liu and Ma (2024) [156] investigated loss oscillations observed in adaptive optimizers, including RMSProp. It explains how RMSProp's exponential moving average mechanism contributes to this phenomenon and proposes a novel perspective on tuning hyperparameters to mitigate oscillations. Li (2024) [157] explored the fundamental theoretical properties of adaptive optimizers, with a special focus on RMSProp. It rigorously examines the interplay between smoothness conditions and the adaptive nature of RMSProp, showing how it balances stability and convergence speed. Heredia (2024) [158] presented a new mathematical framework for analyzing RMSProp using integro-differential equations. The model provides deeper theoretical insights into how RMSProp updates gradients differently from AdaGrad and Adam, particularly in terms of gradient smoothing. Ye (2024) [159] discussed how preconditioning methods, including RMSProp, enhance gradient descent optimization. It explains why RMSProp's adaptive learning rate is beneficial in high-dimensional settings and provides a theoretical justification for its effectiveness in regularized optimization problems. Compagnoni et. al. (2024) [160] employed stochastic differential equations (SDEs) to model the behavior of RMSProp and other adaptive optimizers. It provides new theoretical insights into how noise affects the optimization process and how RMSProp adapts to different gradient landscapes. Yao et. al. (2024) [161] presented a system response curve analysis of first-order optimization methods, including RMSProp. The authors develop a dynamic equation for RMSProp that explains its stability and effectiveness in deep learning tasks. Wen and Lei (2024) [162] explored an alternative optimization framework that integrates RMSProp-style updates with an ADMM approach. It provides theoretical guarantees for the convergence of RMSProp in non-convex optimization problems. Hannibal et. al. (2024) [163] critiques the convergence properties of popular optimizers, including RMSProp. It rigorously proves that in certain settings, RMSProp may not lead to a global minimum, emphasizing the importance of hyperparameter tuning. Yang (2025) [164] extended the theoretical understanding of adaptive optimizers like RMSProp by analyzing the impact of bias in stochastic gradient updates. It provides a rigorous mathematical treatment of how bias affects convergence.

The Root Mean Squared Propagation (RMSProp) optimizer is a sophisticated variant of the gradient descent algorithm that adapts the learning rate for each parameter in a non-linear, non-convex optimization problem. The fundamental issue with standard gradient descent lies in the constant learning rate η, which fails to account for the varying magnitudes of the gradients in different directions of the parameter space. This lack of adaptation can cause inefficient optimization, where large gradients may lead to overshooting and small gradients lead to slow convergence. RMSProp addresses this problem by dynamically adjusting the learning rate based on the historical gradient magnitudes, offering a more tailored and efficient approach. Consider the objective function $f(\theta)$, where $\theta \in \mathbb{R}^n$ is the vector of parameters that we aim to optimize. Let $\nabla f(\theta)$ denote the gradient of $f(\theta)$ with respect to θ, which is a vector of partial derivatives:

$$\nabla f(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1}, \frac{\partial f(\theta)}{\partial \theta_2}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]^T . \tag{15.103}$$

In traditional gradient descent, the update rule for θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t), \tag{15.104}$$

where η is the learning rate, a scalar constant. However, this approach does not account for the fact that the gradient magnitudes may differ significantly along different directions in the parameter space, especially in high-dimensional, non-convex functions. The RMSProp optimizer introduces a solution by adapting the learning rate for each parameter in proportion to the magnitude of the historical gradients. The key modification in RMSProp is the introduction of a running average of the squared gradients for each parameter θ_i, denoted as $E[g^2]_{i,t}$, which captures the cumulative magnitude of the gradients over time. The update rule for $E[g^2]_{i,t}$ is given by the exponential moving average formula:

$$E[g^2]_{i,t} = \beta E[g^2]_{i,t-1} + (1-\beta) g_{i,t}^2, \tag{15.105}$$

where $g_{i,t} = \frac{\partial f(\theta_t)}{\partial \theta_i}$ is the gradient of the objective function with respect to the parameter θ_i at time step t, and β is the decay factor, typically set close to 1 (e.g., $\beta = 0.9$). This recurrence relation allows the gradient history to influence the current update while exponentially forgetting older gradient information. The value of β determines the memory of the squared gradients, where higher values of β give more weight to past gradients. The update for θ_i in RMSProp is then given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\eta}{\sqrt{E[g^2]_{i,t} + \epsilon}} g_{i,t}, \tag{15.106}$$

where ϵ is a small positive constant (typically $\epsilon = 10^{-8}$) introduced to avoid division by zero and ensure numerical stability. The term $\frac{1}{\sqrt{E[g^2]_{i,t+\epsilon}}}$ dynamically adjusts the learning rate for each parameter based on the magnitude of the squared gradient history. This adjustment allows RMSProp to take larger steps in directions where gradients have historically been small, and smaller steps in directions where gradients have been large, leading to a more stable and efficient optimization process. RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm that incorporates the following recursive update for the mean squared gradient:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2. \tag{15.107}$$

where v_t represents the exponentially weighted moving average of squared gradients at time t, $\beta \in (0, 1)$ is the decay rate that determines how much past gradients contribute, $g_t = \nabla_\theta f(\theta_t)$ is the stochastic gradient of the loss function f, g_t^2 represents the **element-wise** squared gradient. The step update for parameters θ is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t. \tag{15.108}$$

where η is the learning rate, and ϵ is a small positive constant for numerical stability. The key term of interest is the **mean squared gradient estimate** v_t, and its mathematical properties will now be studied in extreme rigor. Note that the recurrence equation is

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2 \tag{15.109}$$

can be expanded iteratively:

$$v_t = \beta(\beta v_{t-2} + (1 - \beta) g_{t-1}^2) + (1 - \beta) g_t^2. \tag{15.110}$$

$$= \beta^2 v_{t-2} + (1 - \beta)\beta g_{t-1}^2 + (1 - \beta) g_t^2. \tag{15.111}$$

Continuing this expansion:

$$v_t = \beta^t v_0 + (1 - \beta) \sum_{k=0}^{t-1} \beta^k g_{t-k}^2. \tag{15.112}$$

For sufficiently large t, assuming $v_0 \approx 0$, we obtain:

$$v_t = (1 - \beta) \sum_{k=0}^{t-1} \beta^k g_{t-k}^2. \tag{15.113}$$

which represents an exponentially weighted moving average of past squared gradients. To analyze the expectation, we formally introduce a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ where Ω is the sample space, $\mathcal{F}$ is the sigma-algebra of measurable events, $\mathbb{P}$ is the probability measure governing the stochastic process g_t. The stochastic gradients g_t are assumed to be random variables:

$$g_t : \Omega \to \mathbb{R}^d \tag{15.114}$$

with a well-defined second moment:

$$\mathbb{E}[g_t^2] = \sigma_g^2. \tag{15.115}$$

Applying expectation to both sides of the recurrence:

$$\mathbb{E}[v_t] = (1-\beta)\sum_{k=0}^{t-1} \beta^k \mathbb{E}[g_{t-k}^2]. \tag{15.116}$$

For independent and identically distributed (i.i.d.) gradients:

$$\mathbb{E}[g_t^2] = \sigma_g^2 \quad \forall t. \tag{15.117}$$

Thus:

$$\mathbb{E}[v_t] = (1-\beta)\sigma_g^2 \sum_{k=0}^{t-1} \beta^k. \tag{15.118}$$

Using the closed-form geometric sum:

$$\sum_{k=0}^{t-1} \beta^k = \frac{1-\beta^t}{1-\beta}, \tag{15.119}$$

we obtain:

$$\mathbb{E}[v_t] = \sigma_g^2(1-\beta^t). \tag{15.120}$$

To find the asymptotic Limit, we have to take the limit as $t \to \infty$:

$$\lim_{t\to\infty} \mathbb{E}[v_t] = \sigma_g^2. \tag{15.121}$$

Thus, the mean square estimate **converges to the true second moment of the gradient**. To establish almost sure convergence, consider:

$$v_t - \sigma_g^2 = (1-\beta)\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2). \tag{15.122}$$

By the strong law of large numbers, for a sufficiently large number of iterations:

$$\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2) \to 0 \quad \text{a.s.} \tag{15.123}$$

which implies:

$$v_t \to \sigma_g^2 \quad \text{a.s.} \tag{15.124}$$

In conclusion, the properties of the Mean Square Estimate are

- v_t is a **biased estimator** of σ_g^2 for finite t, but **unbiased in the limit**.
- v_t converges to σ_g^2 **in expectation, variance, and almost surely**.
- This ensures **stable and adaptive learning rates** in RMSprop.

To eliminate bias in early iterations, we define the **bias-adjusted estimate** as:

$$\hat{v}_t = \frac{v_t}{1-\beta^t}. \tag{15.125}$$

This ensures an **unbiased estimation** of the expected squared gradient. The parameter update for RMSprop is as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} g_t. \tag{15.126}$$

where η is the learning rate and ϵ ensures numerical stability. To derive the Bias Correction. We rigorously derive the expected value of v_t using full expansion.

$$\mathbb{E}[v_t] = \mathbb{E}[\beta v_{t-1} + (1-\beta)g_t^2]. \tag{15.127}$$

Applying linearity of expectation:

$$\mathbb{E}[v_t] = \beta\mathbb{E}[v_{t-1}] + (1-\beta)\mathbb{E}[g_t^2]. \tag{15.128}$$

Expanding recursively:

$$\mathbb{E}[v_t] = \beta^t v_0 + (1-\beta)\sum_{k=0}^{t-1} \beta^k \mathbb{E}[g_{t-k}^2]. \tag{15.129}$$

Assuming g_t is an **unbiased estimate** with variance σ_g^2, we get:

$$\mathbb{E}[v_t] = \sigma_g^2(1-\beta^t). \tag{15.130}$$

Since v_t is biased, we correct the expectation by normalizing:

$$\hat{v}_t = \frac{v_t}{1-\beta^t}. \tag{15.131}$$

Thus, the bias-corrected expectation satisfies:

$$\mathbb{E}[\hat{v}_t] = \sigma_g^2. \tag{15.132}$$

This confirms that **bias-adjusted RMSprop provides an unbiased estimate of the second moment**. We now do the Almost Sure Convergence Analysis. For that we analyze convergence by considering the difference:

$$v_t - \sigma_g^2 = (1-\beta)\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2). \tag{15.133}$$

Using the Strong Law of Large Numbers (SLLN):

$$\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2) \to 0 \quad \text{almost surely.} \tag{15.134}$$

Thus,

$$v_t \to \sigma_g^2 \quad \text{a.s.,} \quad \hat{v}_t \to \sigma_g^2 \quad \text{a.s.} \tag{15.135}$$

confirming that **Bias-Adjusted RMSprop provides an asymptotically unbiased estimate** of σ_g^2. Let's do the Stability Analysis of Learning Rate. The **effective learning rate** in RMSprop is:

$$\eta_{\text{eff}} = \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}. \tag{15.136}$$

Therefore we have:

1. Without Bias Correction: If β^t is large in early iterations, then:

   $$v_t \approx (1-\beta)g_t^2. \tag{15.137}$$

   Since $(1-\beta)g_t^2 \ll \sigma_g^2$, the denominator in η_{eff} is **too small**, leading to **excessively large steps**, causing instability.

2. With Bias Correction: Since $\hat{v}_t \to \sigma_g^2$, we ensure that:

   $$\eta_{\text{eff}} \approx \frac{\eta}{\sqrt{\sigma_g^2 + \epsilon}} \tag{15.138}$$

   resulting in **stable step sizes** and improved convergence.

In conclusion, the Mathematical Properties of Bias-Adjusted RMSprop are:

- **Bias correction ensures** $\mathbb{E}[\hat{v}_t] = \sigma_g^2$, removing underestimation.
- **Almost sure convergence guarantees** asymptotically stable second-moment estimation.
- **Stable step sizes prevent instability in early iterations**.

Thus, **Bias-Adjusted RMSprop mathematically improves the stability and convergence behavior of RMSprop**.

Mathematically, the key advantage of RMSProp over traditional gradient descent lies in its ability to adapt the learning rate according to the local geometry of the objective function. In regions where the objective function is steep (large gradients), RMSProp reduces the effective learning rate by dividing by $\sqrt{E[g^2]_{i,t}}$, mitigating the risk of overshooting. Conversely, in flatter regions with smaller gradients, RMSProp increases the learning rate, allowing for faster convergence. This self-adjusting mechanism is crucial in high-dimensional optimization tasks, where the gradients along different directions can vary greatly in magnitude, as is often the case in deep learning tasks involving neural networks. The exponential moving average of squared gradients used in RMSProp is analogous to a form of local normalization, where each parameter is scaled by the inverse of the running average of its gradient squared. This normalization ensures that the optimizer does not become overly sensitive to gradients in any particular direction, thus stabilizing the optimization process. In more formal terms, if the objective function $f(\theta)$ exhibits sharp curvatures along certain directions, RMSProp mitigates the effects of such curvatures by scaling down the step size along those directions. This scaling behavior can be interpreted as a form of gradient re-weighting, where the influence of each parameter's gradient is modulated by its historical behavior, making the optimizer more robust to ill-conditioned optimization problems. The introduction of ϵ ensures that the denominator never becomes zero, even in the case where the squared gradient history for a parameter θ_i becomes extremely small. This is crucial for maintaining the numerical stability of the algorithm, particularly in scenarios where gradients may vanish or grow exceedingly small over many iterations, as seen in certain deep learning applications, such as training very deep neural networks. By providing a small non-zero lower bound to the learning rate, ϵ ensures that the updates remain smooth and predictable.

RMSProp's performance is heavily influenced by the choice of β, which controls the trade-off between long-term history and recent gradient information. When β is close to 1, the optimizer relies more heavily on the historical gradients, which is useful for capturing long-term trends in the optimization landscape. On the other hand, smaller values of β allow the optimizer to be more responsive to recent gradient changes, which can be beneficial in highly non-stationary environments or rapidly changing optimization landscapes. In the context of deep learning, RMSProp is particularly effective for optimizing objective functions with complex, high-dimensional parameter spaces, such as those encountered in training deep neural networks. The non-convexity of such objective functions often leads to a gradient that can vary significantly in magnitude across different layers of the network. RMSProp helps to balance the updates across these layers by adjusting the learning rate based on the historical gradients, ensuring that all layers receive appropriate updates without being dominated by large gradients from any single layer. This adaptability helps in preventing gradient explosions or vanishing gradients, which are common issues in deep learning optimization. In summary, RMSProp provides a robust and efficient optimization technique by adapting the learning rate based on the historical squared gradients of each parameter. The exponential decay of the squared gradient history allows RMSProp to strike a balance between stability and adaptability, preventing overshooting and promoting faster convergence in non-convex optimization problems. The introduction of ϵ ensures numerical stability, and the parameter β offers flexibility in controlling the influence of past gradients. This makes RMSProp particularly well-suited for high-dimensional optimization tasks, especially in deep learning applications, where the parameter space is vast, and gradient magnitudes can differ significantly across dimensions. By effectively normalizing the gradients and dynamically adjusting the learning rates, RMSProp significantly enhances the efficiency and stability of gradient-based optimization methods.

15.3 Overfitting and Regularization Techniques

Literature Review: Goodfellow (2016) et. al. [112] provides a comprehensive introduction to deep learning, including a thorough discussion on overfitting and regularization techniques. It explains methods such as L1/L2 regularization, dropout, batch normalization, and data augmentation, which help improve generalization. The authors explore the bias-variance tradeoff and practical solutions to reduce overfitting in neural networks. Hastie et. al. (2009) [129] discusses overfitting in statistical learning models, particularly in regression and classification. The book covers regularization techniques like Ridge Regression (L2) and Lasso (L1), as well as cross-validation techniques for preventing overfitting. It is fundamental for understanding model complexity control in machine learning. Bishop (2006) [115] in his book provided an in-depth mathematical foundation of machine learning models, with particular attention to regularization methods such as Bayesian inference, early stopping, and weight decay. It emphasized probabilistic interpretations of regularization, demonstrating how overfitting can be mitigated through prior distributions in Bayesian models. Murphy (2012) [130] in his book presents a Bayesian approach to machine learning, covering regularization techniques from a probabilistic viewpoint. It discusses penalization methods, Bayesian regression, and variational inference as tools to control model complexity and prevent overfitting. The book is useful for those looking to understand uncertainty estimation in ML models. Srivastava et. al. (2014) [131] introduced Dropout, a widely used regularization technique in deep learning. The authors show how randomly dropping units during training reduces co-adaptation of neurons, thereby enhancing model generalization. This technique remains a key part of modern neural network training pipelines. Zou and Hastie (2005) [132] introduced Elastic Net, a combination of L1 (Lasso) and L2 (Ridge) regularization, which addresses the limitations of Lasso in handling correlated features. It is particularly useful for high-dimensional data, where feature selection and regularization are crucial. Vapnik (1995) [133] in his introduced Statistical Learning Theory and the VC-dimension, which quantifies model complexity. It provides the mathematical framework explaining why overfitting occurs and how regularization constraints reduce generalization error. It forms the theoretical basis of Support Vector Machines (SVMs) and Structural Risk Minimization. Ng (2004) [134] compares L1 (Lasso) and L2 (Ridge) regularization, demonstrating their impact on feature selection and model stability. It shows that L1 regularization is more effective for sparse models, whereas L2 preserves information better in highly correlated feature spaces. This work is essential for choosing the right regularization technique for specific datasets. Li (2025) [135] explored regularization techniques in high-dimensional clinical trial data using ensemble methods, Bayesian optimization, and deep learning regularization techniques. It highlights the practical application of regularization to prevent overfitting in medical AI. Yasuda (2025) [136] focused on regularization in hybrid machine learning models, specifically Gaussian–Discrete RBMs. It extends L1/L2 penalties and dropout strategies to improve the generalization of deep generative models. It's valuable for those working on deep learning architectures and unsupervised learning.

Overfitting in neural networks is a critical issue where the model learns to excessively fit the training data, capturing not just the true underlying patterns but also the noise and anomalies present in the data. This leads to poor generalization to unseen data, resulting in a model that has a low training error but a high test error. Mathematically, consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the input feature vector for each data point, and $y_i \in \mathbb{R}$ represents the corresponding target value. The goal is to fit a neural network model $f(\mathbf{x}; \mathbf{w})$ parameterized by weights $\mathbf{w} \in \mathbb{R}^M$, where M denotes the number of parameters in the model. The model's objective is to minimize the empirical risk, given by the mean squared error between the predicted values and the true target values:

$$\hat{R}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i; \mathbf{w}), y_i) \tag{15.139}$$

where L denotes the loss function, typically the squared error $L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$. In this framework, the neural network tries to minimize the empirical risk on the training set. However, the true goal is to minimize the expected risk $R(\mathbf{w})$, which reflects the model's performance on the true distribution $P(\mathbf{x}, y)$ of the data. This expected risk is given by:

$$R(\mathbf{w}) = \mathbb{E}_{\mathbf{x},y}[L(f(\mathbf{x}; \mathbf{w}), y)] \tag{15.140}$$

Overfitting occurs when the model minimizes $\hat{R}(\mathbf{w})$ to an excessively small value, but $R(\mathbf{w})$ remains large, indicating that the model has fit the noise in the training data, rather than capturing the true data distribution. This discrepancy arises from an overly complex model that learns to memorize the training data rather than generalizing across different inputs. A fundamental insight into the overfitting phenomenon comes from the bias-variance decomposition of the generalization error. The total error in a model's prediction $\hat{f}(\mathbf{x})$ of the true target function $g(\mathbf{x})$ can be decomposed as:

$$\mathcal{E} = \mathbb{E}[(g(\mathbf{x}) - \hat{f}(\mathbf{x}))^2] = \text{Bias}^2(\hat{f}(\mathbf{x})) + \text{Var}(\hat{f}(\mathbf{x})) + \sigma^2 \tag{15.141}$$

where $\text{Bias}^2(\hat{f}(\mathbf{x}))$ represents the squared difference between the expected model prediction and the true function, $\text{Var}(\hat{f}(\mathbf{x}))$ is the variance of the model's predictions across different training sets, and σ^2 is the irreducible error due to the intrinsic noise in the data. In the context of overfitting, the model typically exhibits low bias (as it fits the training data very well) but high variance (as it is highly sensitive to the fluctuations in the training data). Therefore, regularization techniques aim to reduce the variance of the model while maintaining its ability to capture the true underlying relationships in the data, thereby improving generalization. One of the most popular methods to mitigate overfitting is **L2 regularization** (also known as weight decay), which adds a penalty term to the loss function based on the squared magnitude of the weights. The regularized loss function is given by:

$$\hat{R}_{\text{reg}}(\mathbf{w}) = \hat{R}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2 = \hat{R}(\mathbf{w}) + \lambda\sum_{j=1}^{M} w_j^2 \tag{15.142}$$

where λ is a positive constant controlling the strength of the regularization. The gradient of the regularized loss function with respect to the weights is:

$$\nabla_{\mathbf{w}}\hat{R}_{\text{reg}}(\mathbf{w}) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + 2\lambda\mathbf{w} \tag{15.143}$$

The term $2\lambda\mathbf{w}$ introduces weight shrinkage, which discourages the model from fitting excessively large weights, thus preventing overfitting by reducing the model's complexity. This regularization approach is a direct way to control the model's capacity by penalizing large weight values, leading to a simpler model that generalizes better. In contrast, **L1 regularization** adds a penalty based on the absolute values of the weights:

$$\hat{R}_{\text{reg}}(\mathbf{w}) = \hat{R}(\mathbf{w}) + \lambda\|\mathbf{w}\|_1 = \hat{R}(\mathbf{w}) + \lambda\sum_{j=1}^{M} |w_j| \tag{15.144}$$

The gradient of the L1 regularized loss function is:

$$\nabla_{\mathbf{w}}\hat{R}_{\text{reg}}(\mathbf{w}) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + \lambda\,\text{sgn}(\mathbf{w}) \tag{15.145}$$

where $\text{sgn}(\mathbf{w})$ denotes the element-wise sign function. L1 regularization has a unique property of inducing sparsity in the weights, meaning it drives many of the weights to exactly zero, effectively selecting a subset of the most important features. This feature selection mechanism is particularly useful in high-dimensional settings, where many input features may be irrelevant. A more advanced regularization technique is **dropout**, which randomly deactivates a fraction of neurons during training. Let $\mathbf{h}_i$ represent the activation of the i-th neuron in a given layer. During training, dropout produces a binary mask $\mathbf{m}_i$ sampled from a Bernoulli distribution with success probability p, i.e., $m_i \sim \text{Bernoulli}(p)$, such that:

$$\mathbf{h}_i^{\text{drop}} = \frac{1}{p}\mathbf{m}_i \odot \mathbf{h}_i \tag{15.146}$$

where $\odot$ denotes element-wise multiplication. The factor $1/p$ ensures that the expected value of the activations remains unchanged during training. Dropout effectively forces the network to learn redundant representations, reducing its reliance on specific neurons and promoting better generalization. By training an ensemble of subnetworks with shared weights, dropout helps to prevent the network from memorizing the training data, thus reducing overfitting. **Early stopping** is another technique to prevent overfitting, which involves halting the training process when the validation error starts to increase. The model is trained on the training set, but its performance is evaluated on a separate validation set. If the validation error $R_{\text{val}}(t)$ increases after several epochs, training is stopped to prevent further overfitting. Mathematically, the stopping criterion is:

$$t^* = \arg\min_t R_{\text{val}}(t) \tag{15.147}$$

where t^* represents the epoch at which the validation error reaches its minimum. This technique avoids the risk of continuing to fit the training data beyond the point where the model starts to lose its ability to generalize. **Data augmentation** artificially enlarges the training dataset by applying transformations to the original data. Let $T = \{T_1, T_2, \ldots, T_K\}$ represent a set of transformations (such as rotations, scaling, and translations). For each training example $(\mathbf{x}_i, y_i)$, the augmented dataset $\mathcal{D}'$ consists of K new examples:

$$\mathcal{D}' = \{(T_k(\mathbf{x}_i), y_i) \mid i = 1, 2, \ldots, N, k = 1, 2, \ldots, K\} \tag{15.148}$$

These transformations create new, varied examples, which help the model generalize better by preventing it from fitting too closely to the original, potentially noisy data. Data augmentation is particularly beneficial in domains like image processing, where transformations like rotations and flips do not change the underlying label but provide additional examples to learn from. **Batch normalization** normalizes the activations of each mini-batch to reduce internal covariate shift and stabilize the learning process. Given a mini-batch $\mathcal{B} = \{\mathbf{h}_i\}_{i=1}^m$ with activations $\mathbf{h}_i$, the mean and variance of the activations across the mini-batch are computed as:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^m \mathbf{h}_i, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^m (\mathbf{h}_i - \mu_{\mathcal{B}})^2 \tag{15.149}$$

The normalized activations are then given by:

$$\hat{\mathbf{h}}_i = \frac{\mathbf{h}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{15.150}$$

where ϵ is a small constant for numerical stability. Batch normalization helps to smooth the optimization landscape, allowing for faster convergence and mitigating the risk of overfitting by preventing the model from getting stuck in sharp, narrow minima in the loss landscape.

In conclusion, overfitting is a significant challenge in training neural networks, and its prevention requires a combination of techniques aimed at controlling model complexity, improving generalization, and reducing sensitivity to noise in the training data. Regularization methods such as L2 and L1 regularization, dropout, and early stopping, combined with strategies like data augmentation and batch normalization, are fundamental to improving the performance of neural networks on unseen data and ensuring that they do not overfit the training set. The mathematical formulations and optimization strategies outlined here provide a detailed and rigorous framework for understanding and mitigating overfitting in machine learning models.

15.3.1 Dropout

Literature Review: Srivastava et. al. (2014) [131] introduced dropout as a regularization technique. The authors demonstrated that randomly dropping units (along with their connections) during training prevents overfitting by reducing co-adaptation among neurons. They provided theoretical insights and empirical evidence showing that dropout improves generalization in deep neural networks. Goodfellow et. al. (2016) [112] wrote a comprehensive textbook covers dropout in the context of regularization and overfitting. It explains dropout as an approximate Bayesian inference method and discusses its relationship to ensemble learning and noise injection. The book also provides a broader perspective on regularization techniques in deep learning. Srivastava et. al. (2013) [546] in a technical report expands on the dropout technique, providing additional insights into its implementation and effectiveness. It discusses the impact of dropout on different architectures and datasets, emphasizing its role in reducing overfitting and improving model robustness. Baldi and Sadowski (2013) [547] provided a theoretical analysis of dropout, explaining why it works as a regularization technique. The authors show that dropout can be interpreted as an adaptive regularization method that penalizes large weights, leading to better generalization. While not specifically about dropout, this paper by Zou and Hastie (2005) [132] introduced the Elastic Net, a regularization technique that combines L1 and L2 penalties. It provides foundational insights into regularization methods, which are conceptually related to dropout in their goal of preventing overfitting. Gal and Ghahramani (2016) [548] established a theoretical connection between dropout and Bayesian inference. The authors show that dropout can be interpreted as a variational approximation to a Bayesian neural network, providing a probabilistic framework for understanding its regularization effects. Hastie et. al. (2009) [129] provided a thorough grounding in statistical learning, including regularization techniques. While it predates dropout, it offers essential background on overfitting, bias-variance tradeoff, and regularization methods like ridge regression and Lasso, which are foundational to understanding dropout. Gal et. al. (2016) [549] introduced an improved version of dropout called ”Concrete Dropout” which automatically tunes the dropout rate during training. This innovation addresses the challenge of manually selecting dropout rates and enhances the regularization capabilities of dropout. Gal et. al. (2016) [550] provided a rigorous theoretical analysis of dropout in deep networks. It explores how dropout affects the optimization landscape and the dynamics of training, offering insights into why dropout is effective in preventing overfitting. Friedman et. al. (2010) [551] focused on regularization paths for generalized linear models, emphasizing the importance of regularization in preventing overfitting. While not specific to dropout, it provides a strong foundation for understanding the broader context of regularization techniques in machine learning.

Dropout, a regularization technique in neural networks, is designed to address overfitting, a situation where a model performs well on training data but fails to generalize to unseen data. The general problem of overfitting in machine learning arises when a model becomes excessively complex, with a high number of parameters, and learns to model noise in the data rather than the true underlying patterns. This can result in poor generalization performance on new, unseen data. In the context of neural networks, the solution often involves *regularization techniques* to penalize complexity and prevent the model from memorizing the data. Dropout, introduced by Geoffrey Hinton et al., represents a unique and powerful method to regularize neural networks by introducing stochasticity during the training process, which forces the model to generalize better and prevents overfitting. To understand the mathematics behind dropout, let $f_\theta(x)$ represent the output of a neural network for input x with parameters θ. The goal during training is to minimize a loss function that measures the discrepancy between the predicted output and the true target y. Without any regularization, the objective is to minimize the empirical loss:

$$L_{\text{empirical}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \tag{15.151}$$

where $\mathcal{L}(f_\theta(x_i), y_i)$ is the loss function (e.g., cross-entropy or mean squared error), and N is the number of data samples. A model trained to minimize this loss function without regularization will likely overfit to the training data, capturing the noise rather than the underlying distribution of the data. Dropout addresses this by randomly "dropping out" a fraction of the network's neurons during each training iteration, which is mathematically represented by modifying the activations of neurons.

Let us consider a feedforward neural network with a set of activations a_i for the neurons in the i-th layer, which is computed as $a_i = f(Wx_i + b_i)$, where W represents the weight matrix, x_i the input to the neuron, and b_i the bias. During training with dropout, for each neuron, a random Bernoulli variable r_i is introduced, where:

$$r_i \sim \text{Bernoulli}(p) \tag{15.152}$$

with probability p representing the retention probability (i.e., the probability that a neuron is kept active), and $1 - p$ representing the probability that a neuron is "dropped" (set to zero). The activation of the i-th neuron is then modified as follows:

$$a'_i = r_i \cdot a_i = r_i \cdot f(Wx_i + b_i) \tag{15.153}$$

where r_i is a random binary mask for each neuron. During each forward pass, different neurons are randomly dropped out, and the network is effectively training on a different subnetwork, forcing the network to learn a more robust set of features that do not depend on any particular neuron. In this way, dropout acts as a form of *ensemble learning*, as each forward pass corresponds to a different realization of the network.

The mathematical expectation of the loss function with respect to the dropout mask r can be written as:

$$\mathbb{E}_r[L_{\text{dropout}}(\theta, r)] = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i, r), y_i) \tag{15.154}$$

where $f_\theta(x_i, r)$ is the output of the network with the dropout mask r. Since the dropout mask is random, the loss is an expectation over all possible configurations of dropout masks. This randomness induces an implicit *ensemble effect*, where the model is trained not just on a single set of parameters θ, but effectively on a *distribution* of models, each corresponding to a different dropout configuration. The model is, therefore, regularized because the network is forced to generalize across these different subnetworks, and overfitting to the training data is prevented. One way to gain deeper insight into dropout is to consider its connection with *Bayesian inference*. In the context of deep learning, dropout can be viewed as an approximation to *Bayesian posterior inference*. In Bayesian terms, we seek the posterior distribution of the network's parameters θ, given the data D, which can be written as:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{15.155}$$

where $p(D|\theta)$ is the likelihood of the data given the parameters, $p(\theta)$ is the prior distribution over the parameters, and $p(D)$ is the marginal likelihood of the data. Dropout approximates this posterior by averaging over the outputs of many different subnetworks, each corresponding to a different dropout configuration. This interpretation is formalized by observing that each forward pass with a different dropout mask corresponds to a different realization of the model, and averaging over all dropout masks gives an approximation to the Bayesian posterior. Thus, the expected output of the network, given the data x, under dropout is:

$$\mathbb{E}_r[f_\theta(x)] = \frac{1}{M}\sum_{i=1}^{M} f_\theta(x, r_i) \tag{15.156}$$

where r_i is a dropout mask drawn from the Bernoulli distribution and M is the number of Monte Carlo samples of dropout configurations. This expectation can be interpreted as a form of *ensemble averaging*, where each individual forward pass corresponds to a different model sampled from the posterior.

Dropout is also highly effective because it controls the *bias-variance tradeoff.* The bias-variance tradeoff is a fundamental concept in statistical learning, where increasing model complexity reduces bias but increases variance, and vice versa. A highly complex model tends to have low bias but high variance, meaning it fits the training data very well but fails to generalize to new data. Regularization techniques, such as dropout, seek to reduce variance without increasing bias excessively. Dropout achieves this by introducing stochasticity in the learning process. By randomly deactivating neurons during training, the model is forced to learn *robust features* that do not depend on the presence of specific neurons. In mathematical terms, the variance of the model's output can be expressed as:

$$\mathrm{Var}(f_\theta(x)) = \mathbb{E}_r[(f_\theta(x))^2] - (\mathbb{E}_r[f_\theta(x)])^2 \tag{15.157}$$

By averaging over multiple dropout configurations, the variance is reduced, leading to better generalization performance. Although dropout introduces some bias by reducing the network's capacity (since fewer neurons are available at each step), the variance reduction outweighs the bias increase, resulting in improved generalization. Another key mathematical aspect of dropout is its relationship with *stochastic gradient descent (SGD).* In the standard SGD framework, the parameters θ are updated using the gradient of the loss with respect to the parameters. In the case of dropout, the gradient is computed based on a *stochastic subnetwork* at each training iteration, which introduces an element of randomness into the optimization process. The parameter update rule with dropout can be written as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathbb{E}_r[L_{\text{dropout}}(\theta, r)] \tag{15.158}$$

where η is the learning rate, and ∇_θ is the gradient of the loss with respect to the model parameters. The expectation is taken over all possible dropout configurations, which means that at each step, the gradient update is based on a different realization of the model. This stochasticity helps the optimization process by preventing the model from getting stuck in local minima, improving convergence towards global minima, and enhancing generalization. Finally, it is important to note that dropout has a close connection with *low-rank approximations.* During each forward pass with dropout, certain neurons are effectively removed, which reduces the rank of the weight matrix, as some rows or columns of the matrix are set to zero. This stochastic reduction in rank forces the network to learn lower-dimensional representations of the data, effectively performing *low-rank regularization.* This aspect of dropout can be formalized by observing that each dropout mask corresponds to a sparse matrix, and the network is effectively learning a low-rank approximation of the data distribution. By doing so, dropout prevents the network from learning overly complex representations that could overfit the data, leading to improved generalization.

In summary, dropout is a powerful and mathematically sophisticated regularization technique that introduces randomness into the training process. By randomly deactivating neurons during each forward pass, dropout forces the model to generalize better and prevents overfitting. Dropout can be understood as approximating Bayesian posterior inference over the model parameters and acts as a form of ensemble learning. It controls the bias-variance tradeoff, reduces variance, and improves generalization. The stochastic nature of dropout also introduces a form of noise injection during training, which aids in avoiding local minima and ensures convergence to global minima. Additionally, dropout induces low-rank regularization, which further improves generalization by preventing overly complex representations. Through these mathematical and statistical insights, dropout has become a cornerstone technique in deep learning, enhancing the performance of neural networks on unseen data.

15.3.2 L1/L2 Regularization and Overfitting

Literature Review (L1 (Lasso) Regularization): Hastie et. al. (2009) [129] provided a comprehensive introduction to regularization techniques, including L1 regularization (Lasso). It rigorously explains the bias-variance tradeoff, overfitting, and how L1 regularization induces sparsity in models. The authors also discuss the geometric interpretation of L1 regularization and its application in high-dimensional data. Tibshirani (1996) [552] introduced the Lasso (Least Absolute Shrinkage and Selection Operator). Tibshirani rigorously demonstrates how L1 regularization performs both variable selection and regularization, making it particularly useful for high-dimensional datasets. The paper also provides theoretical insights into the conditions under which Lasso achieves optimal performance. Friedman et. al. (2010) [551] introduced an efficient algorithm for computing the regularization path for L1-regularized generalized linear models (GLMs). It provides a practical framework for implementing L1 regularization in various statistical models, including logistic regression and Poisson regression. Meinshausen (2007) [553] explored the use of L1 regularization for sparse regression and its connection to marginal testing. The authors rigorously analyze the consistency of L1 regularization in high-dimensional settings and provide theoretical guarantees for variable selection. Carvalho. et. al. (2009) [554] extended L1 regularization to Bayesian frameworks, introducing adaptive sparsity-inducing priors. It provides a rigorous Bayesian interpretation of L1 regularization and demonstrates its application in genomics, where overfitting is a significant concern.

Literature Review (L2 (Ridge Regression) Regularization): Hastie et. al. (2009) [129] provided a comprehensive introduction to overfitting and regularization techniques, including L2 regularization. It rigorously explains the bias-variance tradeoff, the mathematical formulation of ridge regression, and its role in controlling model complexity. The book also contrasts L2 regularization with L1 regularization (lasso) and elastic net, offering deep insights into their theoretical and practical implications. Bishop and Nashrabodi (2006) [115] provided a Bayesian perspective on regularization, explaining L2 regularization as a Gaussian prior on model parameters. The book rigorously derives the connection between ridge regression and maximum a posteriori (MAP) estimation, offering a probabilistic interpretation of regularization. Friedman et. al. (2010) [551] introduced efficient algorithms for solving regularized regression problems, including L2 regularization. It provides a detailed analysis of the computational aspects of regularization and its impact on model performance. The authors also discuss the interplay between L2 regularization and other regularization techniques in the context of generalized linear models. Hoerl and Kennard (1970) [555] introduced ridge regression (L2 regularization). The authors demonstrated how adding a small positive constant to the diagonal of the design matrix (ridge penalty) can stabilize the solution of ill-posed regression problems, reducing overfitting and improving generalization. Goodfellow et. al. (2016) [112] provided a modern perspective on regularization in the context of deep learning. It discusses L2 regularization as a method to penalize large weights in neural networks, preventing overfitting. The authors also explore the interaction between L2 regularization and other techniques like dropout and batch normalization. Cesa-Bianchi et.al. (2004) [556] provided a theoretical analysis of the generalization ability of learning algorithms, including those using L2 regularization. It rigorously connects regularization to the concept of Rademacher complexity, offering a framework for understanding how regularization controls overfitting by limiting the complexity of the hypothesis space. Devroye et. al. (2013) [557] provided a rigorous theoretical foundation for understanding overfitting and regularization. It discusses L2 regularization in the context of risk minimization and explores its role in achieving consistent and stable learning algorithms. Zou and Hastie (2005) [132] introduced the elastic net, a hybrid regularization method that combines L1 and L2 penalties. While the focus is on elastic net, the paper provides valuable insights into the properties of L2 regularization, particularly its ability to handle correlated predictors and improve model stability. Abu-Mostafa et. al. (2012) [558] offered an accessible yet rigorous introduction to overfitting and regularization. It explains L2 regularization as a tool to balance fitting the training data and maintaining model simplicity, with clear examples and practical insights. Shalev-Shwartz and Ben-David (2014) [559] provided a theoretical foundation for understanding overfitting and regularization. It rigorously analyzes L2 regularization in the context of empirical risk minimization, highlighting its role in controlling the complexity of linear models and ensuring generalization.

L1 and L2 regularization plays a critical role in mitigating **overfitting**. Overfitting occurs when a model fits not only the underlying data distribution but also the noise in the data, leading to poor generalization to unseen examples. Overfitting is especially prevalent in models with a large number of features, where the model becomes overly flexible and may capture spurious correlations between the features and the target variable. This often results in a model with high variance, where small fluctuations in the data cause significant changes in the model predictions. To combat this, **regularization techniques** are employed, which introduce a penalty term into the objective function, discouraging overly complex models that fit noise.

Given a set of n observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where each $\mathbf{x}_i \in \mathbb{R}^p$ is a feature vector and $y_i \in \mathbb{R}$ is the corresponding target value, the task is to find a parameter vector $\theta \in \mathbb{R}^p$ that minimizes the **loss function**. In standard linear regression, the objective is to minimize the **mean squared error (MSE)**, defined as:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^T\theta\right)^2 = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 \tag{15.159}$$

where $\mathbf{X} \in \mathbb{R}^{n\times p}$ is the design matrix, with rows $\mathbf{x}_i^T$, and $\mathbf{y} \in \mathbb{R}^n$ is the vector of target values. The solution to this problem, without any regularization, is given by the **ordinary least squares (OLS)** solution:

$$\hat{\theta}_{\text{OLS}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{15.160}$$

This formulation, however, can lead to overfitting when p is large or when $\mathbf{X}^T\mathbf{X}$ is nearly singular. In such cases, **regularization** is used to modify the loss function, adding a penalty term $\mathcal{R}(\theta)$ to the objective function that discourages large values for the parameters θ_i. The **regularized loss function** is given by:

$$\mathcal{L}_{\text{regularized}}(\theta) = \mathcal{L}(\theta) + \lambda\mathcal{R}(\theta) \tag{15.161}$$

where λ is a **regularization parameter** that controls the strength of the penalty. The term $\mathcal{R}(\theta)$ penalizes the complexity of the model by imposing constraints on the magnitude of the coefficients. Let us explore two widely used forms of regularization: **L1 regularization** (Lasso) and **L2 regularization** (Ridge). L1 regularization involves adding the **ℓ_1-norm** of the parameter vector θ as the penalty term:

$$\mathcal{R}_{L1}(\theta) = \sum_{i=1}^{p}|\theta_i| \tag{15.162}$$

The corresponding **L1 regularized loss function** is:

$$\mathcal{L}_{L1}(\theta) = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda\sum_{i=1}^{p}|\theta_i| \tag{15.163}$$

This formulation promotes sparsity in the parameter vector θ, causing many coefficients to become exactly zero, effectively performing **feature selection**. In high-dimensional settings where many features are irrelevant, L1 regularization helps reduce the model complexity by forcing irrelevant features to be excluded from the model. The effect of the L1 penalty can be understood geometrically by noting that the constraint region defined by the ℓ_1-norm is a **diamond-shaped** region in p-dimensional space. When solving this optimization problem, the coefficients often lie on the boundary of this diamond, leading to a sparse solution with many coefficients being exactly zero. Mathematically, the **soft-thresholding** solution that arises from solving the L1 regularized optimization problem is given by:

$$\hat{\theta}_i = \operatorname{sign}(\theta_i) \max\left(0, |\theta_i| - \lambda\right) \tag{15.164}$$

This soft-thresholding property drives coefficients to zero when their magnitude is less than λ, resulting in a sparse solution. L2 regularization, on the other hand, uses the **ℓ_2-norm** of the parameter vector θ as the penalty term:

$$\mathcal{R}_{L2}(\theta) = \sum_{i=1}^{p} \theta_i^2 \tag{15.165}$$

The corresponding **L2 regularized loss function** is:

$$\mathcal{L}_{L2}(\theta) = \frac{1}{n} \|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda \sum_{i=1}^{p} \theta_i^2 \tag{15.166}$$

This penalty term does not force any coefficients to be exactly zero but rather **shrinks** the coefficients towards zero, effectively reducing their magnitudes. The L2 regularization helps stabilize the solution when there is multicollinearity in the features by reducing the impact of highly correlated features. The optimization problem with L2 regularization leads to a **ridge regression** solution, which is given by the following expression:

$$\hat{\theta}_{\text{ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1} \mathbf{X}^T\mathbf{y} \tag{15.167}$$

where $\mathbf{I}$ is the identity matrix. The L2 penalty introduces a **circular or spherical** constraint in the parameter space, resulting in a solution where all coefficients are reduced in magnitude, but none are eliminated. The **Elastic Net** regularization is a hybrid technique that combines both L1 and L2 regularization. The regularized loss function for Elastic Net is given by:

$$\mathcal{L}_{\text{ElasticNet}}(\theta) = \frac{1}{n} \|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda_1 \sum_{i=1}^{p} |\theta_i| + \lambda_2 \sum_{i=1}^{p} \theta_i^2 \tag{15.168}$$

In this case, λ_1 and λ_2 control the strength of the L1 and L2 penalties, respectively. The Elastic Net regularization is particularly useful when dealing with datasets where many features are correlated, as it combines the **sparsity-inducing** property of L1 regularization with the **stability-enhancing** property of L2 regularization. The Elastic Net has been shown to outperform L1 and L2 regularization in some cases, particularly when there are groups of correlated features. The optimization problem can be solved using **coordinate descent** or **proximal gradient methods**, which efficiently handle the mixed penalties. The choice of regularization parameter λ is critical in controlling the **bias-variance tradeoff**. A small value of λ leads to a low-penalty model that is more prone to overfitting, while a large value of λ forces the coefficients to shrink towards zero, potentially leading to underfitting. Thus, it is important to select an optimal value for λ to strike a balance between bias and variance. This can be achieved by using **cross-validation** techniques, where the model is trained on a subset of the data, and the performance is evaluated on the remaining data.

In conclusion, both L1 and L2 regularization techniques play an important role in addressing overfitting by controlling the complexity of the model. L1 regularization encourages sparsity and feature selection, while L2 regularization reduces the magnitude of the coefficients without eliminating any features. By incorporating these regularization terms into the objective function, we can achieve a more balanced bias-variance tradeoff, enhancing the model's ability to generalize to new, unseen data.

15.3.3 Elastic Net Regularization

Literature Review: Zou and Hastie (2005) [132] introduced the Elastic Net regularization method. The authors combined the strengths of L1 (Lasso) and L2 (Ridge) regularization to address their individual limitations. Lasso can select only a subset of variables, while Ridge tends to shrink coefficients but does not perform variable selection. Elastic Net balances these by encouraging group selection of correlated variables and improving prediction accuracy, especially when the number of predictors exceeds the number of observations. Hastie et. al. (2010) [129] provided a comprehensive overview of statistical learning methods, including detailed discussions on overfitting, regularization techniques, and the Elastic Net. It explains the theoretical foundations of regularization, the bias-variance tradeoff, and practical implementations of Elastic Net in high-dimensional data settings. Tibshirani (1996) [552] introduced the Lasso (L1 regularization), which is a key component of Elastic Net. Lasso performs both variable selection and regularization by shrinking some coefficients to zero. The paper laid the groundwork for understanding how L1 regularization can prevent overfitting in high-dimensional datasets. Hoerl and Kennard (1970) [555] introduced Ridge Regression (L2 regularization), which addresses multicollinearity and overfitting by shrinking coefficients toward zero without setting them to zero. Ridge Regression is the other key component of Elastic Net, and this paper provides the theoretical basis for its use in regularization. Bühlmann and van de Geer (2011) [560] provided a rigorous treatment of high-dimensional statistics, including regularization techniques like Elastic Net. It discusses the theoretical properties of Elastic Net, such as its ability to handle correlated predictors and its consistency in variable selection. Friedman et. al. (2010) [551] presented efficient algorithms for computing regularization paths for Lasso, Ridge, and Elastic Net in generalized linear models. The authors introduce coordinate descent, a computationally efficient method for fitting Elastic Net models, making it practical for large-scale datasets. Gareth et. al. (2013) [561] provided an accessible introduction to regularization techniques, including Elastic Net. It explains the intuition behind overfitting, the bias-variance tradeoff, and how Elastic Net combines L1 and L2 penalties to improve model performance. Efron et. al. (2004) [562] introduced the Least Angle Regression (LARS) algorithm, which is closely related to Lasso and Elastic Net. LARS provides a computationally efficient way to compute the regularization path for Lasso and Elastic Net, making it easier to understand the behavior of these methods. Fan and Li (2001) [563] discussed the theoretical properties of variable selection methods, including Lasso and Elastic Net. It introduces the concept of oracle properties, which ensure that the selected model performs as well as if the true underlying model were known. The paper provides insights into why Elastic Net is effective in high-dimensional settings. Meinshausen and Bühlmann (2006) [564] explored the use of Lasso and related methods (including Elastic Net) in high-dimensional settings. It provides theoretical guarantees for variable selection consistency and discusses the challenges of overfitting in high-dimensional data. The insights from this paper are directly applicable to understanding the performance of Elastic Net.

Overfitting is a critical issue in machine learning and statistical modeling, where a model learns the training data too well, capturing not only the underlying patterns but also the noise and outliers, leading to poor generalization performance on unseen data. Mathematically, overfitting can be characterized by a significant discrepancy between the training error $E_{\text{train}}(\theta)$ and the test error $E_{\text{test}}(\theta)$, where θ represents the model parameters. Specifically, $E_{\text{train}}(\theta)$ is minimized during training, but $E_{\text{test}}(\theta)$ remains high, indicating that the model has failed to generalize. This typically occurs when the model complexity, quantified by the number of parameters or the degrees of freedom, is excessively high relative to the amount of training data available. To mitigate overfitting, regularization techniques are employed, and among these, Elastic Net regularization stands out as a particularly effective method due to its ability to combine the strengths of both L1 (Lasso) and L2 (Ridge) regularization. Elastic Net regularization addresses overfitting by introducing a penalty term to the loss function that constrains the magnitude of the model parameters θ. The general form of the regularized loss function is given by

$$L(\theta) = \text{Data Loss}(\theta) + \lambda \cdot \text{Penalty}(\theta) \tag{15.169}$$

where λ is the regularization parameter controlling the strength of the penalty, and Penalty(θ) is a function that penalizes large or complex parameter values. In Elastic Net, the penalty term is a convex combination of the L1 and L2 norms of the parameter vector θ, expressed as

$$\text{Penalty}(\theta) = \alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2 \tag{15.170}$$

Here,

$$\|\theta\|_1 = \sum_{i=1}^{n} |\theta_i| \tag{15.171}$$

is the L1 norm, which encourages sparsity by driving some parameters to exactly zero, and

$$\|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2 \tag{15.172}$$

is the squared L2 norm, which discourages large parameter values and promotes smoothness. The mixing parameter $\alpha \in [0, 1]$ controls the balance between the L1 and L2 penalties, with $\alpha = 1$ corresponding to pure Lasso regularization and $\alpha = 0$ corresponding to pure Ridge regularization. For a linear regression model, the Elastic Net loss function takes the form

$$L(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 + \lambda\left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \tag{15.173}$$

where m is the number of training examples, y_i is the target value for the i-th example, x_i is the feature vector for the i-th example, and θ is the vector of model parameters. The first term in the loss function,

$$\frac{1}{2m}\sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 \tag{15.174}$$

represents the mean squared error (MSE) of the model predictions, while the second term,

$$\lambda\left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \tag{15.175}$$

represents the Elastic Net penalty. The regularization parameter λ controls the overall strength of the penalty, with larger values of λ resulting in stronger regularization and simpler models. The optimization problem for Elastic Net regularization is formulated as

$$\min_{\theta} \left\{ \frac{1}{2m}\sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 + \lambda\left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \right\} \tag{15.176}$$

This is a convex optimization problem, and its solution can be obtained using iterative algorithms such as coordinate descent or proximal gradient methods. The coordinate descent algorithm updates one parameter at a time while holding the others fixed, and the update rule for the j-th parameter θ_j is given by

$$\theta_j \leftarrow \frac{S\left(\sum_{i=1}^{m} x_{ij}(y_i - \tilde{y}_i^{(-j)}), \lambda\alpha\right)}{1 + \lambda(1-\alpha)} \tag{15.177}$$

where $S(z, \gamma)$ is the soft-thresholding operator defined as

$$S(z, \gamma) = \text{sign}(z) \max(|z| - \gamma, 0) \tag{15.178}$$

and $\tilde{y}_i^{(-j)}$ is the predicted value excluding the contribution of θ_j. The Elastic Net penalty has several desirable properties that make it particularly effective for overfitting control. First, the L1 component ($\alpha\|\theta\|_1$) induces sparsity in the parameter vector θ, effectively performing feature selection by setting some coefficients to zero. This is especially useful in high-dimensional settings where the number of features n is much larger than the number of training examples m. Second, the L2 component ($(1 - \alpha)\|\theta\|_2^2$) encourages a grouping effect, where correlated features tend to have similar coefficients. Third, the mixing parameter α provides flexibility in balancing the sparsity-inducing effect of L1 regularization with the smoothness-promoting effect of L2 regularization. In practice, the hyperparameters λ and α must be carefully tuned to achieve optimal performance. This is typically done using cross-validation. The Elastic Net regularization path, which describes how the coefficients θ change as λ varies, can be computed efficiently using algorithms such as least angle regression (LARS) with Elastic Net modifications.

In conclusion, Elastic Net regularization is a mathematically rigorous and scientifically sound technique for controlling overfitting in machine learning models. By combining the sparsity-inducing properties of L1 regularization with the smoothness-promoting properties of L2 regularization, Elastic Net provides a flexible and effective framework for handling high-dimensional data, multicollinearity, and feature selection.

15.3.4 Early Stopping

Literature Review: Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including detailed discussions on overfitting and regularization techniques. It explains early stopping as a form of regularization that prevents overfitting by halting training when validation performance plateaus. The book rigorously connects early stopping to other regularization methods like weight decay and dropout, emphasizing its role in controlling model complexity. Montavon et. al. (2012) [565] compiled practical techniques for training neural networks, including early stopping. It highlights how early stopping acts as an implicit regularizer by limiting the effective capacity of the model. The authors provide empirical evidence and theoretical insights into why early stopping works, comparing it to explicit regularization methods like L2 regularization. Bishop (2006) [115] provided a rigorous mathematical treatment of overfitting and regularization. It discusses early stopping in the context of gradient-based optimization, showing how it prevents overfitting by controlling the effective number of parameters. The book also connects early stopping to Bayesian inference, framing it as a way to balance model complexity and data fit. Prechelt (1998) [566] provided a systematic analysis of early stopping criteria, such as generalization loss and progress measures. He introduces quantitative metrics to determine the optimal stopping point and demonstrates its effectiveness in preventing overfitting across various datasets and architectures. Zhang et. al. (2021) [445] challenged traditional views on generalization in deep learning. It shows that deep neural networks can fit random labels, highlighting the importance of regularization techniques like early stopping. The authors argue that early stopping is crucial for ensuring models generalize well, even in the presence of high capacity. Friedman et. al. (2010) [551] introduced coordinate descent algorithms for regularized linear models, including L1 and L2 regularization. While not exclusively about early stopping, it provides a theoretical framework for understanding how regularization techniques, including early stopping, control model complexity and prevent overfitting. Hastie et. al. (2010) [129] discussed early stopping as a regularization method in the context of gradient boosting and neural networks. The authors explain how early stopping reduces variance by limiting the number of iterations, thereby improving generalization performance. While primarily focused on dropout, Srivastava et. al. (2014) [131] compared dropout to other regularization techniques, including early stopping. It highlights how early stopping complements dropout by preventing overfitting during training. The authors provide empirical results showing the combined benefits of these methods

Overfitting in machine learning models is a phenomenon where the model learns to approximate the training data with excessive precision, capturing not only the underlying data-generating distribution but also the noise and stochastic fluctuations inherent in the finite sample of training data. Formally, consider a model $f(x;\theta)$, parameterized by $\theta \in \mathbb{R}^d$, which maps input features $x \in \mathbb{R}^p$ to predictions $\hat{y} \in \mathbb{R}$. The model is trained to minimize the empirical risk $L_{\text{train}}(\theta)$, defined over a training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i are the input features and y_i are the corresponding labels. The empirical risk is given by:

$$L_{\text{train}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{15.179}$$

where $\ell(\cdot)$ is a loss function quantifying the discrepancy between the predicted output $f(x_i;\theta)$ and the true label y_i. Overfitting occurs when the model achieves a very low training loss $L_{\text{train}}(\theta)$ but a significantly higher generalization loss $L_{\text{test}}(\theta)$, evaluated on an independent test dataset D_{test}. This discrepancy arises because the model has effectively memorized the training data, including its noise, rather than learning the true underlying patterns.

Early stopping is a regularization technique that mitigates overfitting by dynamically halting the training process before the model fully converges to a minimum of the training loss. This is achieved by monitoring the model's performance on a separate validation dataset $D_{\text{val}} = \{(x_j, y_j)\}_{j=1}^{M}$, which is distinct from both the training and test datasets. The validation loss $L_{\text{val}}(\theta)$ is computed as:

$$L_{\text{val}}(\theta) = \frac{1}{M}\sum_{j=1}^{M} \ell(f(x_j;\theta), y_j) \tag{15.180}$$

During training, the model parameters θ are updated iteratively using an optimization algorithm such as gradient descent, which follows the update rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L_{\text{train}}(\theta_t) \tag{15.181}$$

where η is the learning rate and $\nabla_\theta L_{\text{train}}(\theta_t)$ is the gradient of the training loss with respect to the parameters θ at iteration t. Early stopping intervenes in this process by evaluating the validation loss $L_{\text{val}}(\theta_t)$ at each iteration t and terminating training when $L_{\text{val}}(\theta_t)$ ceases to decrease or begins to increase. This point of termination is determined by a patience parameter P, which specifies the number of iterations to wait after the last improvement in $L_{\text{val}}(\theta_t)$ before stopping. The effectiveness of early stopping as a regularization mechanism can be understood through its implicit control over the model's complexity. By limiting the number of training iterations T, early stopping restricts the model's capacity to fit the training data perfectly, thereby preventing it from overfitting. This can be formalized by considering the relationship between the number of iterations T and the effective complexity of the model. Specifically, early stopping imposes an implicit constraint on the optimization process, preventing the model from reaching a sharp minimum of the training loss $L_{\text{train}}(\theta)$, which is often associated with poor generalization. Instead, early stopping encourages convergence to a flatter minimum, which is more robust to perturbations in the data. The regularization effect of early stopping can be further analyzed through its connection to explicit regularization techniques. It has been shown that early stopping is mathematically equivalent to imposing an implicit L_2 regularization penalty on the model parameters θ. This equivalence arises because early stopping effectively restricts the norm of the parameter updates $\|\theta_t - \theta_0\|$, where θ_0 is the initial parameter vector. The strength of this implicit regularization is inversely proportional to the number of iterations T, as fewer iterations result in smaller updates to θ. Formally, this can be expressed as:

$$\|\theta_T - \theta_0\| \leq C(T) \tag{15.182}$$

where $C(T)$ is a function that decreases with T. This constraint on the parameter updates is analogous to the explicit L_2 regularization penalty $\lambda\|\theta\|_2^2$, where λ controls the strength of the regularization. Thus, early stopping can be viewed as a form of adaptive regularization, where the regularization strength is determined by the number of iterations T. The theoretical foundation of early stopping is further supported by its connection to the bias-variance tradeoff in statistical learning. By limiting the number of iterations T, early stopping reduces the variance of the model, as it prevents the model from fitting the noise in the training data. At the same time, it introduces a small amount of bias, as the model may not fully capture the underlying data-generating distribution. This tradeoff is optimized by selecting the stopping point T that minimizes the generalization error, which can be estimated using cross-validation or a held-out validation set.

In summary, early stopping is a powerful and theoretically grounded technique for controlling overfitting in machine learning models. By dynamically halting the training process based on the validation loss, it imposes an implicit regularization constraint on the model parameters, preventing them from growing too large and overfitting the training data. This regularization effect is mathematically equivalent to an implicit L_2 penalty, and it is rooted in the principles of optimization theory and statistical learning. Through its connection to the bias-variance tradeoff, early stopping provides a principled approach to balancing model complexity and generalization performance, making it an essential tool in the machine learning practitioner's toolkit.

15.3.5 Data Augmentation

Literature Review: Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including detailed discussions on overfitting and regularization techniques. It explains how data augmentation acts as a form of regularization by introducing variability into the training data, thereby reducing the model's reliance on specific patterns and improving generalization. The book also covers other regularization methods like dropout, weight decay, and early stopping, contextualizing their relationship with data augmentation. Zou and Hastie (2005) [132] introduced the Elastic Net, a regularization technique that combines L1 (Lasso) and L2 (Ridge) penalties. While not directly about data augmentation, it provides a theoretical foundation for understanding how regularization combats overfitting. The principles discussed are highly relevant when designing augmentation strategies to ensure that models do not overfit to augmented data. Zhang et. al. (2021) [445] challenged traditional notions of generalization in deep learning. It demonstrates that deep neural networks can easily fit random labels, highlighting the importance of regularization techniques, including data augmentation, to prevent overfitting. The study underscores the role of augmentation in improving generalization by making the learning task more challenging and robust. Srivastava et. al. (2014) [131] introduced dropout, a regularization technique that randomly deactivates neurons during training. While the focus is on dropout, the authors discuss how data augmentation complements dropout by providing additional training examples, thereby further reducing overfitting. The paper provides empirical evidence of the synergy between augmentation and dropout. Brownlee (2019) [567] focused on implementing data augmentation techniques for image data using popular deep learning frameworks. It provides a hands-on explanation of how augmentation reduces overfitting by increasing the diversity of training data. The book also discusses the interplay between augmentation and other regularization methods like weight decay and batch normalization. Shorten and Khoshgoftaar (2019) [569] provided a comprehensive review of data augmentation techniques across various domains, including images, text, and audio. It rigorously analyzes how augmentation serves as a regularization mechanism by introducing noise and variability into the training process, thereby preventing overfitting. The paper also discusses the limitations and challenges of augmentation. Friedman et. al. (2010) [551] introduced efficient algorithms for fitting regularized generalized linear models. While primarily focused on L1 and L2 regularization, it provides insights into how regularization techniques can be combined with data augmentation to control model complexity and prevent overfitting. The paper is particularly useful for understanding the theoretical underpinnings of regularization. Zhang et. al. (2017) [568] introduced Mixup, a data augmentation technique that creates new training examples by linearly interpolating between pairs of inputs and their labels. Mixup acts as a form of regularization by encouraging the model to behave linearly between training examples, thereby reducing overfitting. The paper provides theoretical and empirical evidence of its effectiveness. Cubuk et al. (2019) [571] proposed AutoAugment, a method for automatically learning optimal data augmentation policies from data. By tailoring augmentation strategies to the specific dataset, AutoAugment acts as a powerful regularization technique, significantly reducing overfitting and improving model performance. The paper demonstrates the effectiveness of this approach on multiple benchmarks. Perez (2017) [570] provided a detailed empirical study of how data augmentation reduces overfitting in deep neural networks. It compares various augmentation techniques and their impact on model generalization. The authors also discuss the relationship between augmentation and other regularization methods, providing insights into how they can be combined for optimal performance.

Overfitting, in its most formal and rigorous definition, arises when a model $f \in \mathcal{H}$, where $\mathcal{H}$ denotes the hypothesis space of all possible models, achieves a low empirical risk on the training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$ but fails to generalize to unseen data drawn from the true data-generating distribution P. This phenomenon can be quantified by the discrepancy between the model's performance on the training data and its performance on the test data, which can be expressed mathematically as:

$$E_{(x,y)\sim P}[L(\hat{f}(x), y)] \gg \frac{1}{N} \sum_{i=1}^{N} L(\hat{f}(x_i), y_i) \tag{15.183}$$

where L is the loss function measuring the error between the model's predictions $\hat{f}(x)$ and the true labels y, and $\hat{f}$ is the model that minimizes the empirical risk on D_{train}. The primary cause of overfitting is the model's excessive capacity to fit the training data, which is often a consequence of high model complexity relative to the size and diversity of D_{train}. Data augmentation addresses overfitting by artificially expanding the training dataset D_{train} through the application of a set of transformations $\mathcal{T}$ to the existing data points. These transformations are designed to preserve the semantic content of the data while introducing variability that reflects plausible real-world variations. Formally, let $T : X \to X$ be a transformation function that maps an input $x \in X$ to a transformed input $T(x)$. The augmented dataset D_{aug} is then constructed as:

$$D_{\text{aug}} = \{(T(x_i), y_i) \mid x_i \in D_{\text{train}}, T \in \mathcal{T}\}. \tag{15.184}$$

The model is subsequently trained on D_{aug} instead of D_{train}, which effectively increases the size of the training dataset and introduces additional diversity. This process can be viewed as implicitly defining a new empirical risk minimization problem:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \frac{1}{|D_{\text{aug}}|} \sum_{(x_i, y_i) \in D_{\text{aug}}} L(f(x_i), y_i). \tag{15.185}$$

By training on D_{aug}, the model is exposed to a broader range of data variations, which encourages it to learn more robust and generalizable features. This reduces the risk of overfitting by preventing the model from over-relying on specific patterns or noise present in the original training data. The effectiveness of data augmentation can be analyzed through the lens of the bias-variance trade-off. Without data augmentation, the model may exhibit high variance due to its ability to fit the limited training data too closely. Data augmentation reduces this variance by effectively increasing the size of the training dataset, thereby constraining the model's capacity to fit noise. At the same time, it introduces a controlled form of bias by encouraging the model to learn features that are invariant to the applied transformations. This trade-off can be formalized by considering the expected generalization error E_{gen} of the model, which decomposes into bias and variance terms:

$$E_{\text{gen}} = E_{(x,y)\sim P}\left[(\hat{f}(x) - y)^2\right] = \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \sigma^2, \tag{15.186}$$

where σ^2 represents the irreducible noise in the data. Data augmentation reduces $\text{Var}(\hat{f})$ by increasing the effective sample size, while the bias term $\text{Bias}(\hat{f})$ may increase slightly due to the constraints imposed by the invariance requirements. The choice of transformations $\mathcal{T}$ is critical to the success of data augmentation. For instance, in image classification tasks, common transformations include rotations, translations, scaling, flipping, and color jittering. Each transformation $T \in \mathcal{T}$ can be represented as a function $T : \mathbb{R}^d \to \mathbb{R}^d$, where d is the dimensionality of the input space. The set $\mathcal{T}$ should be designed such that the transformed data points $T(x)$ remain semantically consistent with the original labels y. Mathematically, this can be expressed as:

$$P(y \mid T(x)) \approx P(y \mid x) \quad \forall T \in \mathcal{T}. \tag{15.187}$$

This ensures that the augmented data points are valid representatives of the underlying data distribution P. In addition to reducing overfitting, data augmentation also has the effect of smoothing the loss landscape of the optimization problem. The loss function L evaluated on the augmented dataset D_{aug} can be viewed as a regularized version of the original loss function:

$$L_{\text{aug}}(f) = \frac{1}{|D_{\text{aug}}|} \sum_{(x_i,y_i)\in D_{\text{aug}}} L(f(x_i), y_i). \tag{15.188}$$

This augmented loss function typically exhibits a more convex and smoother optimization landscape, which facilitates convergence during training. The smoothness of the loss landscape can be quantified using the Lipschitz constant L of the gradient ∇L_{aug}, which satisfies:

$$\|\nabla L_{\text{aug}}(f_1) - \nabla L_{\text{aug}}(f_2)\| \leq L\|f_1 - f_2\| \quad \forall f_1, f_2 \in \mathcal{H}. \tag{15.189}$$

A smaller Lipschitz constant L indicates a smoother loss landscape, which is beneficial for optimization algorithms such as gradient descent.

In conclusion, data augmentation is a powerful and mathematically grounded technique for controlling overfitting in machine learning models. By artificially expanding the training dataset through the application of semantically preserving transformations, data augmentation reduces the model's reliance on specific patterns and noise in the original training data. This leads to improved generalization performance by balancing the bias-variance trade-off and smoothing the optimization landscape. The rigorous formulation of data augmentation as a form of implicit regularization provides a solid theoretical foundation for its widespread use in practice.

15.3.6 Cross-Validation

Literature Review: Hastie et. al. (2010) [129] provided a comprehensive overview of statistical learning methods, including detailed discussions on overfitting, bias-variance tradeoff, and regularization techniques (e.g., Ridge Regression, Lasso). It also covers cross-validation as a tool for model selection and evaluation. The book rigorously explains how regularization mitigates overfitting by introducing penalty terms to the loss function, and how cross-validation helps in tuning hyperparameters. Tibshirani (1996) [552] introduced the Lasso (Least Absolute Shrinkage and Selection Operator), a regularization technique that performs both variable selection and shrinkage to prevent overfitting. The paper demonstrates how Lasso's L1 penalty encourages sparsity in model coefficients, making it particularly useful for high-dimensional data. It also discusses cross-validation for selecting the regularization parameter. Bishop and Nashrabodi (2006) [115] provided a deep dive into probabilistic models and regularization techniques, including Bayesian regularization and weight decay. It explains how regularization controls model complexity and prevents overfitting by penalizing large weights. The book also discusses cross-validation as a method for assessing model performance and selecting hyperparameters. Hoerl and Kennard (1970) [555] introduced Ridge Regression, an L2 regularization technique that addresses multicollinearity and overfitting in linear models. The authors demonstrate how adding a penalty term to the least squares objective function shrinks coefficients, reducing variance at the cost of introducing bias. Cross-validation is highlighted as a method for choosing the optimal regularization parameter. Domingos (2012) [572] provided practical insights into machine learning, including the importance of avoiding overfitting and the role of regularization. He emphasized the tradeoff between model complexity and generalization, and how techniques like cross-validation help in selecting models that generalize well to unseen data. Goodfellow et. al. (2016) [112] covered regularization techniques specific to deep learning, such as dropout, weight decay, and early stopping. It explains how these methods prevent overfitting in neural networks and discusses cross-validation as a tool for hyperparameter tuning. The book also explores the theoretical underpinnings of regularization in the context of deep models. Srivastava et. al. (2014) [131] introduced dropout, a regularization technique for neural networks that randomly deactivates neurons during training. The authors demonstrate that dropout reduces overfitting by preventing co-adaptation of neurons and effectively ensembles multiple sub-networks. Cross-validation is used to evaluate the performance of dropout-regularized models. Gareth et. al. (2013) [561] provided an accessible introduction to key concepts in statistical learning, including overfitting, regularization, and cross-validation. It explains how techniques like Ridge Regression and Lasso improve model generalization and how cross-validation helps in selecting the best model. The book includes practical examples and R code for implementation. Stone (1974) [573] formalized the concept of cross-validation as a method for assessing predictive performance and preventing overfitting. Stone discusses how cross-validation provides an unbiased estimate of model performance by partitioning data into training and validation sets. The paper lays the groundwork for using cross-validation in conjunction with regularization techniques. Friedman et. al. (2010) [551] presented efficient algorithms for computing regularization paths for generalized linear models, including Lasso and Elastic Net. The authors demonstrate how these techniques balance bias and variance to prevent overfitting. The paper also discusses the use of cross-validation for selecting the optimal regularization parameters.

Overfitting in supervised learning is fundamentally characterized by a learned function f that exhibits low training error but high generalization error. Mathematically, this is framed through the concept of expected risk minimization. Given a probability distribution $P(x,y)$ over the feature-label space, the goal of supervised learning is to minimize the expected risk functional:

$$R(f) = \mathbb{E}_{(x,y)\sim P}[L(y, f(x))] \tag{15.190}$$

where $L(y, f(x))$ is a loss function measuring the discrepancy between predicted and actual values. Since $P(x,y)$ is unknown, we approximate $R(f)$ with the empirical risk over the training dataset $D = \{(x_i, y_i)\}_{i=1}^N$, yielding the empirical risk functional:

$$\hat{R}(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) \tag{15.191}$$

A model is said to overfit if there exists another model g such that $\hat{R}(f) < \hat{R}(g)$ but $R(f) > R(g)$. This discrepancy is analytically understood through the bias-variance decomposition of the generalization error:

$$\mathbb{E}[(y - f(x))^2] = (\mathbb{E}[f(x)] - f^*(x))^2 + V[f(x)] + \sigma^2 \tag{15.192}$$

Overfitting corresponds to the regime where $V[f(x)]$ is significantly large while $(\mathbb{E}[f(x)] - f^*(x))^2$ remains small, meaning that the model is highly sensitive to variations in the training set. Cross-validation provides a principled mechanism for estimating $R(f)$ and preventing overfitting by simulating model performance on unseen data. The most rigorous formulation of cross-validation is k-fold cross-validation, where the dataset D is partitioned into k disjoint subsets $D_1, D_2, \dots, D_k$, each containing approximately $\frac{N}{k}$ samples. For each $j \in \{1, 2, \dots, k\}$, we train the model on the dataset

$$D_{\text{train}}(j) = D \setminus D_j \tag{15.193}$$

and evaluate it on the validation set D_j, computing the validation error:

$$\hat{R}^j(f) = \frac{1}{|D_j|} \sum_{(x_i,y_i)\in D_j} L(y_i, f(x_i)) \tag{15.194}$$

The cross-validation estimate of the expected risk is given by:

$$\hat{R}_{\text{CV}}(f) = \frac{1}{k} \sum_{j=1}^{k} \hat{R}^j(f) \tag{15.195}$$

This estimation introduces a tradeoff between bias and variance depending on the choice of k. A small k, such as $k = 2$, results in high bias due to insufficient training data per fold, while large k, such as $k = N$ (leave-one-out cross-validation, LOOCV), results in high variance due to the extreme sensitivity of the validation error to single observations. The variance of the cross-validation estimator itself is approximated by:

$$\text{Var}(\hat{R}_{\text{CV}}) = \frac{1}{k} \sum_{j=1}^{k} \text{Var}(\hat{R}^j) \tag{15.196}$$

Leave-one-out cross-validation is particularly insightful as it provides an almost unbiased estimate of $R(f)$. Formally, if $D_{-i} = D \setminus \{(x_i, y_i)\}$, then the leave-one-out estimator is:

$$\hat{R}_{\text{LOO}}(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_{-i}(x_i)) \tag{15.197}$$

where f_{-i} is the model trained on D_{-i}. The key advantage of LOOCV is its nearly unbiased nature,

$$\mathbb{E}[\hat{R}_{\text{LOO}}] \approx R(f) \tag{15.198}$$

but its computational cost scales as $O(N)$ times the cost of training the model, making it infeasible for large datasets. Another important mathematical consequence of cross-validation is its role in hyperparameter selection. Suppose a model f_λ is parameterized by λ (e.g., the regularization parameter in Ridge regression). Cross-validation allows us to find

$$\lambda^* = \arg\min_{\lambda} \hat{R}_{\text{CV}}(f_\lambda) \tag{15.199}$$

This optimization ensures that the selected hyperparameter minimizes generalization error rather than just empirical risk. In practical applications, hyperparameter tuning via cross-validation is often performed over a logarithmic grid $\{\lambda_1, \lambda_2, \dots, \lambda_m\}$, and the optimal λ^* is obtained via

$$\lambda^* = \arg\min_{\lambda_j} \frac{1}{k} \sum_{j=1}^{k} \hat{R}^j(f_{\lambda_j}) \tag{15.200}$$

This selection mechanism rigorously prevents overfitting by ensuring that the model complexity is chosen based on its generalization capacity rather than its fit to the training data. A deeper understanding of the bias-variance tradeoff in cross-validation is achieved through its impact on model complexity. If $f_d(x)$ denotes a model of complexity d, its cross-validation risk behaves as:

$$R_{\text{CV}}(f_d) = R(f_d) + O\left(\frac{d}{N}\right) \tag{15.201}$$

This formulation makes explicit that increasing model complexity d leads to lower empirical risk but higher variance, necessitating cross-validation as a control mechanism to balance these competing factors. Finally, an advanced theoretical justification for cross-validation arises from stability theory. The stability of a learning algorithm quantifies how small perturbations in the training set affect its output. Formally, a learning algorithm is γ-stable if, for two datasets D and D' differing by a single point

$$\sup_x |f_D(x) - f_{D'}(x)| \leq \gamma \tag{15.202}$$

Cross-validation is most effective for stable algorithms, where γ-stability ensures that

$$\left|\hat{R}_{CV} - R(f)\right| = O(\gamma) \tag{15.203}$$

For highly unstable algorithms (e.g., deep neural networks with small datasets), cross-validation estimates exhibit significant variance, making regularization even more critical.

In conclusion, cross-validation provides a mathematically rigorous framework for controlling overfitting by estimating generalization error. By partitioning the dataset into training and validation sets, it enables optimal hyperparameter selection and model assessment while managing the bias-variance tradeoff. The interplay between cross-validation risk, model complexity, and stability theory underpins its fundamental role in statistical learning.

15.3.7 Pruning

Literature Review: LeCun et. al. (1990) [574] introduced the concept of pruning in neural networks. They proposed the "optimal brain damage" (OBD) and "optimal brain surgeon" (OBS) algorithms, which prune weights based on their contribution to the loss function. These techniques reduce overfitting by simplifying the model architecture. They proved that pruning based on second-order derivatives (Hessian matrix) is more effective than random pruning, as it preserves critical weights. Li et. al. (2016) [575] focused on pruning convolutional neural networks (CNNs) by removing entire filters rather than individual weights. It demonstrates that filter pruning significantly reduces computational cost while maintaining accuracy, effectively addressing overfitting in large CNNs. The Pruning filters based on their L1-norm magnitude is a simple yet effective regularization technique. Frankle and Carbin (2018) [576] introduced the "lottery ticket hypothesis," which states that dense neural networks contain smaller subnetworks ("winning tickets") that, when trained in isolation, achieve comparable performance to the original network. Pruning helps identify these subnetworks, reducing overfitting by focusing on essential parameters. The authors proposed that Iterative pruning and retraining can uncover sparse, highly generalizable models. Han et. al. (2015) [577] proposed a pruning technique that removes redundant connections and retrains the network to recover accuracy. It introduces a systematic approach to pruning and demonstrates its effectiveness in reducing overfitting while compressing models. The authors proposed that Pruning followed by retraining preserves model performance and reduces overfitting by eliminating unnecessary complexity. Liu et. al. (2018) [578] challenged the conventional wisdom that pruning is primarily for model compression. It shows that pruning can also serve as a regularization technique, improving generalization by removing redundant parameters. The authors proposed that Pruning can be viewed as a form of architecture search, leading to models that generalize better. Cheng et. al. (2017) [579] provided a comprehensive overview of model compression techniques, including pruning, quantization, and knowledge distillation. It highlights how pruning reduces overfitting by simplifying models and removing redundant parameters. The authors proposed that Pruning is a key component of a broader strategy to improve model efficiency and generalization. Frankle et. al. (2020) [580] investigated the limitations of pruning neural networks at initialization (before training). It highlights the challenges of identifying important weights early and suggests that iterative pruning during training is more effective for regularization. The authors proposed that Pruning is most effective when combined with training, as it allows the model to adapt to the reduced architecture.

Overfitting is a core problem in statistical learning theory, occurring when a model exhibits a disproportionately high variance relative to its bias, leading to poor generalization. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^N$ drawn from an unknown probability distribution $P(X, Y)$, a neural network function $f(X, W)$ parameterized by weights W aims to approximate the true underlying function $g(X)$. The goal is to minimize the **true risk function**:

$$R(W) = \mathbb{E}_{(X,Y)\sim P}[\ell(f(X, W), Y)] \tag{15.204}$$

where $\ell(\cdot, \cdot)$ is a chosen loss function such as mean squared error or cross-entropy. Since $P(X, Y)$ is unknown, we approximate $R(W)$ by minimizing the **empirical risk**:

$$\hat{R}(W) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i, W), y_i) \tag{15.205}$$

If W has too many parameters, the model can memorize training data, leading to an excessive gap between the empirical and true risk:

$$R(W) = \hat{R}(W) + \mathcal{O}\left(\sqrt{\frac{d_{\text{VC}}}{N}}\right) \tag{15.206}$$

where d_{VC} is the **Vapnik-Chervonenkis (VC) dimension**, a fundamental measure of model complexity. Overfitting occurs when d_{VC} is excessively large relative to N, leading to high variance. **Pruning** aims to **reduce d_{VC} while preserving network functionality**, thereby controlling complexity and improving generalization. The Mathematical Formulation of Pruning is of a Constrained Optimization Problem. Pruning can be rigorously formulated as a **constrained empirical risk minimization** problem. The objective is to minimize the empirical risk while enforcing a constraint on the number of nonzero weights. Mathematically, this is expressed as:

$$\min_W \hat{R}(W) \quad \text{subject to} \quad \|W\|_0 \leq k \tag{15.207}$$

where $\|W\|_0$ is the **L0 norm**, counting the number of nonzero parameters, and k is the sparsity constraint. Since direct L0 minimization is computationally intractable (NP-hard), practical approaches approximate this problem using **continuous relaxations** such as L1 regularization or thresholding heuristics.

Let's now discuss some theoretical Justifications of different Types of Pruning. For Weight Pruning we start with eliminating Redundant Parameters. Weight pruning removes individual weights that contribute negligibly to the network's predictions. Given a weight matrix W, the simplest form of pruning is threshold-based removal:

$$W' = \{w_j \in W \mid |w_j| > \tau\} \tag{15.208}$$

This operation enforces an **L0-like sparsity constraint**:

$$\Omega(W) = \sum_{j=1}^{d} \mathbf{1}_{|w_j|>\tau} \tag{15.209}$$

Since **direct L0 minimization is non-differentiable**, a common alternative is **L1 regularization**:

$$\hat{W} = \arg\min_{W} \left[\hat{R}(W) + \lambda \sum_{j=1}^{d} |w_j| \right] \tag{15.210}$$

L1 pruning results in a **soft-thresholding effect**, where small weights decay towards zero, reducing model complexity in a **continuous and differentiable** manner. Neuron Pruning is defined as the removing of entire neurons based on activation strength. Beyond individual weights, entire neurons can be pruned based on their average activation magnitude. Given a neuron $h_i(x)$ in layer l with weight vector W_i, we define its **mean absolute activation** over the dataset as:

$$A_i = \frac{1}{N} \sum_{j=1}^{N} |h_i(x_j)|. \tag{15.211}$$

If $A_i < \tau$, then neuron h_i is removed. This corresponds to the minimization:

$$\Omega(W) = \sum_{i=1}^{m} \mathbf{1}_{A_i > \tau}. \tag{15.212}$$

Neuron pruning leads to a direct reduction in **network depth**, modifying the function class and affecting expressivity. The **effective VC dimension** of a fully connected network of depth L with layer sizes $\{n_1, n_2, \ldots, n_L\}$ satisfies:

$$d_{\mathrm{VC}} \approx \sum_{l=1}^{L} n_l^2. \tag{15.213}$$

After pruning p percent of neurons, the new VC dimension is:

$$d'_{\mathrm{VC}} = \sum_{l=1}^{L} (1-p)^2 n_l^2. \tag{15.214}$$

Since generalization error is bounded as $O(\sqrt{d_{\mathrm{VC}}/N})$, reducing d_{VC} via pruning improves generalization. In convolutional networks, structured pruning eliminates entire filters rather than individual weights. Let $F_1, F_2, \ldots, F_m$ be the filters of a convolutional layer. The **importance of filter** F_i is quantified by its **Frobenius norm**:

$$\|F_i\|_F = \sqrt{\sum_{j,k} F_{i,j,k}^2}. \tag{15.215}$$

Filters with norms below threshold τ are removed, solving the optimization problem:

$$\hat{F} = \arg\min_{F} \left[\hat{R}(F) + \lambda \sum_{i=1}^{m} \|F_i\|_F \right] \tag{15.216}$$

Pruning filters leads to **significant reductions in computational cost**, directly improving inference speed while maintaining accuracy. There are some Generalization Bounds for Pruned Networks: PAC Learning and VC Dimension Reduction. A pruned neural network exhibits a **reduced function class complexity**, leading to stronger generalization guarantees. The **PAC (Probably Approximately Correct) bound** states that for any confidence level δ, the probability of excessive generalization error is bounded by:

$$P\left(R(W') - \hat{R}(W') > \epsilon \right) \le 2 \exp\left(-\frac{2N\epsilon^2}{d'_{\mathrm{VC}}} \right) \tag{15.217}$$

Since pruning reduces d'_{VC}, it results in a **tighter PAC bound**, enhancing model robustness. In conclusion, Pruning is an **extremely scientifically and mathematically rigorous** approach to overfitting control, rooted in **optimization theory, PAC learning, VC dimension reduction, and empirical risk minimization**. By removing redundant weights, neurons, or filters, pruning **improves generalization**, **tightens complexity bounds**, and **enhances computational efficiency**.

15.3.8 Ensemble Methods

Literature Review: Hastie et. al. (2009) [129] provided a comprehensive overview of ensemble methods, including bagging, boosting, and random forests. It rigorously explains how overfitting occurs in ensemble models and discusses regularization techniques such as shrinkage in boosting (e.g., AdaBoost, gradient boosting) and feature subsampling in random forests. The book also introduces the bias-variance tradeoff, which is central to understanding overfitting in ensemble methods. Breiman (1996) [581] introduced bagging (Bootstrap Aggregating), an ensemble technique that reduces overfitting by averaging predictions from multiple models trained on bootstrapped samples. The paper demonstrates how bagging reduces variance without increasing bias, making it a powerful regularization tool for unstable models like decision trees. Breiman (2001) [582] introduced random forests, an extension of bagging that further reduces overfitting by introducing randomness in feature selection during tree construction. Breiman shows how random forests achieve regularization through feature subsampling and ensemble averaging, making them robust to overfitting while maintaining high predictive accuracy. Freund and Schapire (1997) [583] introduced AdaBoost, a boosting algorithm that combines weak learners into a strong ensemble. The authors discuss how boosting can overfit noisy datasets and propose theoretical insights into controlling overfitting through careful weighting of training examples and early stopping. Friedman (2001) [584] introduced gradient boosting machines (GBM), a powerful ensemble method that generalizes boosting to differentiable loss functions. The paper emphasizes the importance of shrinkage (learning rate) as a regularization technique to control overfitting. It also discusses the role of tree depth and subsampling in improving generalization. Zhou (2025) [585] provided a systematic and theoretical treatment of ensemble methods, including detailed discussions on overfitting and regularization. It covers techniques such as diversity promotion in ensembles, weighted averaging, and regularized boosting, offering insights into how these methods mitigate overfitting. Dietterich (2000) [586] empirically compared bagging, boosting, and randomization techniques for constructing ensembles of decision trees. It highlights how each method addresses overfitting, with a focus on the role of randomization in reducing model variance and improving generalization. Chen and Guestrin (2016) [587] introduced XGBoost, a highly efficient and scalable implementation of gradient boosting. XGBoost incorporates several regularization techniques, including L1/L2 regularization on leaf weights, column subsampling, and shrinkage, to control overfitting. The paper also discusses the importance of early stopping and cross-validation in preventing overfitting. Bühlmann and Yu (2003) [588] explored boosting with the L2 loss function and its regularization properties. The authors demonstrate how boosting with L2 loss naturally incorporates shrinkage and early stopping as mechanisms to prevent overfitting, providing theoretical guarantees for its generalization performance. While not exclusively focused on ensemble methods, the paper by Snoek et. al. (2012) [489] introduced Bayesian optimization as a tool for hyperparameter tuning in machine learning models, including ensembles. It highlights how optimizing regularization parameters (e.g., learning rate, subsampling rate) can mitigate overfitting and improve ensemble performance.

Overfitting in ensemble methods arises when a model learns the specific noise in the training data rather than capturing the underlying data distribution. Mathematically, given an i.i.d. dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \mathbb{R}$ (for regression) or $y_i \in \{0, 1\}$ (for classification), we consider a hypothesis space $\mathcal{H}$ containing functions $f : \mathbb{R}^d \to \mathbb{R}$ that approximate the true function $f^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}]$. The generalization ability of a model is characterized by its

true risk, defined as

$$\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}}[\ell(f(\mathbf{x}), y)] \tag{15.218}$$

where $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ is the loss function. However, since the true distribution $\mathbb{P}(\mathbf{x}, y)$ is unknown, we approximate this risk using the **empirical risk**,

$$\hat{\mathcal{R}}(f) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i). \tag{15.219}$$

Overfitting occurs when the empirical risk is minimized at the cost of a large true risk, i.e.,

$$\hat{\mathcal{R}}(f) \ll \mathcal{R}(f), \tag{15.220}$$

which leads to poor generalization. This phenomenon can be rigorously analyzed using the **bias-variance decomposition**, which states that the expected squared error of a learned function f satisfies

$$\mathbb{E}[(f(\mathbf{x}) - y)^2] = (\mathbb{E}[f(\mathbf{x})] - f^*(\mathbf{x}))^2 + \mathbb{V}[f(\mathbf{x})] + \sigma^2. \tag{15.221}$$

The first term represents the **bias**, which measures systematic deviation from the true function. The second term represents the **variance**, which quantifies the sensitivity of f to fluctuations in the training data. The third term, σ^2, represents irreducible noise inherent in the data. Overfitting occurs when the variance term dominates, which is particularly problematic in ensemble methods when base learners are highly complex. To understand overfitting in **boosting**, consider a sequence of models $f_1, f_2, \dots, f_T$ iteratively trained to correct errors of previous models. The boosting procedure constructs a final model as a weighted sum:

$$F_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}). \tag{15.222}$$

For **AdaBoost**, the weights α_t are chosen to minimize the exponential loss,

$$\mathcal{L}(F_T) = \sum_{i=1}^{N} \exp(-y_i F_T(\mathbf{x}_i)). \tag{15.223}$$

Differentiating with respect to F_T, we obtain the gradient update rule

$$\nabla_{F_T} \mathcal{L} = -\sum_{i=1}^{N} y_i \exp(-y_i F_T(\mathbf{x}_i)), \tag{15.224}$$

which shows that boosting places **exponentially increasing emphasis on misclassified points**, leading to overfitting when noise is present in the data. For **bagging**, which constructs multiple base models f_m trained on bootstrap samples and aggregates their predictions as

$$F(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\mathbf{x}), \tag{15.225}$$

we analyze variance reduction. If f_m are independent with variance σ^2, then the ensemble variance satisfies

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2. \tag{15.226}$$

However, in practice, base models are correlated, introducing a term ρ such that

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2 + \left(1 - \frac{1}{M}\right) \rho\sigma^2. \tag{15.227}$$

As $M \to \infty$, variance reduction is limited by ρ, which is exacerbated when deep decision trees are used, leading to overfitting. To combat overfitting, **regularization** techniques are employed. One approach is **pruning** in decision trees, where complexity is controlled by minimizing

$$\mathcal{L}(T) = \sum_{i=1}^{N} \ell(f_T(\mathbf{x}_i), y_i) + \lambda |T|, \tag{15.228}$$

where $|T|$ is the number of terminal nodes, and λ penalizes complexity. Another approach is **shrinkage in boosting**, where the update rule is modified to

$$F_{t+1}(\mathbf{x}) = F_t(\mathbf{x}) + \eta h_t(\mathbf{x}), \tag{15.229}$$

where η is a step size satisfying $0 < \eta < 1$. Theoretical analysis shows that small η ensures the ensemble function sequence remains in a Lipschitz-continuous function space, preventing overfitting. Finally, in **random forests**, overfitting is mitigated by decorrelating base models through **feature subsampling**. Given a feature set $\mathcal{F}$ of dimension d, each base tree is trained on a randomly selected subset $\mathcal{F}_m \subset \mathcal{F}$ of size $k \ll d$, ensuring models remain diverse. Theoretical analysis shows that feature selection reduces expected correlation ρ between base models, thereby decreasing ensemble variance:

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2 + \left(1 - \frac{1}{M}\right) \frac{k}{d}\sigma^2. \tag{15.230}$$

Thus, by rigorously analyzing bias-variance tradeoffs, deriving variance-reduction formulas, and proving shrinkage effectiveness, we ensure ensemble methods generalize effectively.

15.3.9 Noise Injection

Literature Review: Hinton and Van Camp (1993) [589] did an early exploration of weight noise as a regularization mechanism. It formalizes the idea that injecting Gaussian noise into neural network weights reduces model complexity, prevents overfitting, and improves interpretability. Bishop (1995) [590] laid the foundation for using noise injection as a regularization method. The paper mathematically formalizes how noise can act as a stochastic approximation of weight decay and discusses its effects on model stability and generalization. Grandvalet and Bengio (2005) [591] explored the use of label noise and entropy minimization for improving model generalization. It demonstrates that adding noise to labels, rather than inputs or weights, can effectively reduce overfitting in semi-supervised learning scenarios. Wager et. al. (2013) [592] offers a theoretical analysis of dropout as a noise-driven adaptive regularization method. It provides a connection between dropout and ridge regression, demonstrating how it acts as a form of adaptive weight scaling to mitigate overfitting. Srivastava et. al. (2014) [131] formally introduced dropout as a regularization technique, showing how randomly omitting neurons during training simulates noise injection and prevents co-adaptation of units. It presents extensive experiments proving that dropout improves test accuracy and generalization. Gal and Ghahramani (2015) [548] extended the concept of dropout by linking it to Bayesian inference, arguing that dropout noise serves as an implicit prior distribution that controls overfitting. It provides rigorous theoretical justifications and empirical studies supporting the role of noise-based regularization in deep learning. Pei et. al. (2025) [593] explored the application of noise injection techniques in convolutional neural networks (CNNs) for electric vehicle load forecasting. It investigates the impact of different regularization methods, including L1/L2 penalties, dropout, and Gaussian noise injection, on reducing overfitting. The study highlights how controlled noise perturbations can enhance generalization performance in time-series forecasting tasks. Chen (2024) [594] demonstrated how noise injection, combined with data augmentation techniques like rotation and shifting, serves as an implicit regularization technique in deep learning models. The study finds that while noise injection marginally improves AUC scores, its effect varies depending on the complexity of the dataset, making it a viable yet context-dependent method for controlling overfitting. An et. al. (2024) [595] introduced a noise-based regularized cross-entropy (RCE) loss function for robust brain tumor segmentation. It argues that controlled noise injection during training prevents overfitting by making models less sensitive to small variations in input data. The study provides empirical evidence that noise-assisted learning improves segmentation performance by enhancing feature robustness. Song and Liu (2024) [596] presented a novel adversarial training technique integrating label noise as a form of regularization. It investigates the theoretical underpinnings of noise injection in preventing catastrophic overfitting in adversarial settings and provides a comparative analysis with traditional dropout and weight decay methods.

Overfitting arises when a model $\hat{f}(x;\theta)$, parameterized by $\theta \in \Theta$, learns not only the true underlying function $f(x) = \mathbb{E}[Y|X=x]$ but also the noise $\epsilon = Y - f(X)$ present in the training data $D = \{(x_i, y_i)\}_{i=1}^n$. Formally, the generalization error $E_{\text{gen}}(\theta)$ and training error $E_{\text{train}}(\theta)$ are defined as:

$$E_{\text{gen}}(\theta) = \mathbb{E}_{(X,Y)\sim P}\left[L(Y, \hat{f}(X;\theta))\right], \tag{15.231}$$

$$E_{\text{train}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i;\theta)) \tag{15.232}$$

where L is a loss function. Overfitting occurs when $E_{\text{gen}}(\theta) \gg E_{\text{train}}(\theta)$, indicating that the model has high variance and poor generalization. This phenomenon is exacerbated when the hypothesis class Θ has excessive capacity, as measured by its Vapnik-Chervonenkis (VC) dimension or Rademacher complexity. Regularization addresses overfitting by introducing a penalty term $R(\theta)$ to the empirical risk minimization problem:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\left(\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i;\theta)) + \lambda \cdot R(\theta)\right) \tag{15.233}$$

where $\lambda > 0$ is a hyperparameter controlling the trade-off between fitting the data and minimizing the penalty. Common choices for $R(\theta)$ include the ℓ_2-norm $\|\theta\|_2^2$ (ridge regression) and the ℓ_1-norm $\|\theta\|_1$ (lasso). These penalties constrain the model's capacity, favoring solutions with smaller norms and reducing variance. Noise injection is a stochastic regularization technique that introduces randomness into the training process to improve generalization. For input noise injection, let $\eta \sim Q$ be a random noise vector sampled from a distribution Q (e.g., Gaussian $N(0, \sigma^2 I)$). The perturbed input is $\tilde{x}_i = x_i + \eta_i$, and the modified training objective becomes:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\eta_i\sim Q}\left[L(y_i, \hat{f}(\tilde{x}_i;\theta))\right]. \tag{15.234}$$

This expectation can be approximated using Monte Carlo sampling or analyzed using a second-order Taylor expansion:

$$\mathbb{E}_{\eta}\left[L(y_i, \hat{f}(x_i+\eta;\theta))\right] \approx L(y_i, \hat{f}(x_i;\theta)) + \frac{\sigma^2}{2}\text{Tr}\left(\nabla_x^2 L(y_i, \hat{f}(x_i;\theta))\right), \tag{15.235}$$

where $\nabla_x^2 L$ is the Hessian matrix of the loss with respect to the input. The second term acts as an implicit regularizer, penalizing the curvature of the loss function and encouraging smoother solutions. For weight noise injection, noise is added directly to the model parameters: $\tilde{\theta} = \theta + \eta$, where $\eta \sim Q$. The training objective becomes:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\eta\sim Q}\left[L(y_i, \hat{f}(x_i;\tilde{\theta}))\right]. \tag{15.236}$$

This formulation encourages the model to converge to flatter minima in the loss landscape, which are associated with better generalization. The flatness of a minimum can be quantified using the eigenvalues of the Hessian matrix $\nabla_\theta^2 L$. Output noise injection introduces randomness into the target labels: $\tilde{y}_i = y_i + \epsilon_i$, where $\epsilon_i \sim Q$. The training objective becomes:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\epsilon_i \sim Q} \left[L(\tilde{y}_i, \hat{f}(x_i; \theta)) \right]. \tag{15.237}$$

This prevents the model from fitting the training labels too closely, reducing overfitting and improving robustness. Theoretical guarantees for noise injection can be derived using tools from statistical learning theory. The Rademacher complexity of the hypothesis class Θ is reduced by noise injection, leading to tighter generalization bounds. The empirical Rademacher complexity is defined as:

$$\hat{R}_n(\Theta) = \mathbb{E}_\sigma \left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \hat{f}(x_i; \theta) \right], \tag{15.238}$$

where σ_i are Rademacher random variables. Noise injection effectively reduces $\hat{R}_n(\Theta)$, as the model is forced to learn robust features that are invariant to small perturbations. From a PAC-Bayesian perspective, noise injection can be interpreted as a form of distributional robustness. It ensures that the model performs well not only on the training distribution but also on perturbed versions of it. The PAC-Bayesian bound takes the form:

$$\mathbb{E}_{\theta \sim Q}\left[E_{\text{gen}}(\theta)\right] \leq \mathbb{E}_{\theta \sim Q}\left[E_{\text{train}}(\theta)\right] + \frac{1}{2n}\text{KL}(Q \| P) + \frac{\log n}{\delta 2n}, \tag{15.239}$$

where Q is the posterior distribution over parameters induced by noise injection, P is a prior distribution, and $\text{KL}(Q\|P)$ is the Kullback-Leibler divergence. In the continuous-time limit, noise injection can be modeled as a stochastic differential equation (SDE):

$$d\theta_t = -\nabla_\theta L(\theta_t) dt + \sigma dW_t, \tag{15.240}$$

where W_t is a Wiener process. This SDE converges to a stationary distribution that favors flat minima, which generalize better. The stationary distribution $p(\theta)$ satisfies the Fokker-Planck equation:

$$\nabla_\theta \cdot \big(p(\theta) \nabla_\theta L(\theta)\big) + \frac{\sigma^2}{2} \nabla_\theta^2 p(\theta) = 0. \tag{15.241}$$

The flatness of the minima can be quantified using the eigenvalues of the Hessian matrix $\nabla_\theta^2 L$. From an information-theoretic perspective, noise injection increases the entropy of the model's predictions, reducing overconfidence and improving calibration. The mutual information $I(\theta; D)$ between the parameters and the data is reduced, leading to better generalization. The information bottleneck principle formalizes this intuition:

$$\min_\theta I(\theta; D) \quad \text{subject to} \quad \mathbb{E}_{(X,Y) \sim P}\big[L(Y, \hat{f}(X; \theta))\big] \leq \epsilon, \tag{15.242}$$

where ϵ is a tolerance parameter. In conclusion, noise injection is a mathematically rigorous and theoretically grounded regularization technique that enhances generalization by introducing controlled stochasticity into the training process. Its effects can be precisely analyzed using tools from functional analysis, stochastic processes, and statistical learning theory, making it a powerful tool for combating overfitting in machine learning models.

15.3.10 Batch Normalization

Literature Review: Cakmakci (2024) [598] explored the use of batch normalization and regularization to improve prediction accuracy in deep learning models. It discusses how BN stabilizes gradients and reduces covariate shifts, preventing overfitting. It also evaluates different combinations of dropout and weight regularization for optimizing performance in pediatric bone age estimation. Surana et. al. (2024) [599] applied dropout regularization and batch normalization in deep learning models for weather forecasting. It provides empirical evidence on how BN prevents overfitting by normalizing inputs at each layer, ensuring smooth training and avoiding the vanishing gradient problem. Chanda (2025) [600] explored the role of batch normalization and dropout in image classification tasks. It highlights how BN maintains the stability of activations, while dropout introduces stochasticity to prevent overfitting in large-scale datasets. Zaitoon et. al. (2024) [601] presented a hybrid regularization approach combining spatial dropout and batch normalization. The authors show how batch normalization smooths feature distributions, leading to faster convergence, while dropout enhances model generalization in GAN-based survival prediction models. Bansal et. al. (2024) [602] integrated Gaussian noise, dropout, and batch normalization to develop a robust fall detection system. It provides a comparative analysis of different regularization methods and highlights how batch normalization helps maintain generalization even in noisy environments. Kusumaningtyas et. al. (2024) [603] investigated batch normalization as a core regularization method in CNN architectures, particularly MobileNetV2. It emphasized how BN reduces internal covariate shift, leading to faster training and better generalization. Hosseini et. al. (2025) [597] applied batch normalization and dropout techniques in medical image classification. It demonstrates that batch normalization stabilizes activations while dropout prevents model dependency on specific neurons, enhancing robustness. Yadav et. al. (2024) [604] examined batch normalization combined with ReLU activations in medical imaging applications. The authors show that batch normalization speeds up convergence and reduces overfitting, leading to more accurate segmentation in cancer detection. Alshamrani and Alshomran (2024) [605] implemented batch normalization along with L2 regularization in ResNet50-based mammogram classification. It highlights how BN reduces parameter sensitivity, improving stability and reducing overfitting in deep learning architectures. Zamindar (2024) [606] applied batch normalization and early stopping techniques in industrial AI applications. It presents an in-depth analysis of how BN prevents overfitting by maintaining variance stability, ensuring improved feature learning.

Overfitting, in its most rigorous formulation, arises when a model $f(x;\theta)$, parameterized by θ, achieves a low empirical risk

$$\hat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{15.243}$$

on the training data $D = \{(x_i, y_i)\}_{i=1}^{N}$, but a high expected risk

$$R(\theta) = \mathbb{E}_{(x,y)\sim P}\left[\ell(f(x;\theta), y)\right] \tag{15.244}$$

on the true data distribution $P(x, y)$. This discrepancy is quantified by the generalization gap

$$R(\theta) - \hat{R}(\theta), \tag{15.245}$$

which can be bounded using tools from statistical learning theory, such as the Rademacher complexity $R_N(H)$ of the hypothesis space H. Specifically, with probability at least $1-\delta$, the generalization gap satisfies:

$$R(\theta) - \hat{R}(\theta) \leq 2R_N(H) + \frac{1}{2N}\log\left(\frac{1}{\delta}\right), \tag{15.246}$$

where

$$R_N(H) = \mathbb{E}_{D,\sigma}\left[\sup_{f\in H}\frac{1}{N}\sum_{i=1}^{N}\sigma_i f(x_i)\right], \tag{15.247}$$

and σ_i are Rademacher random variables. Overfitting occurs when the model complexity, as measured by $R_N(H)$, is too large relative to the sample size N, leading to a high generalization gap. Regularization addresses overfitting by introducing a penalty term $\Omega(\theta)$ into the empirical risk minimization framework, yielding the regularized loss function:

$$L_{\text{reg}}(\theta) = \hat{R}(\theta) + \lambda\Omega(\theta), \tag{15.248}$$

where λ controls the strength of regularization. Common choices for $\Omega(\theta)$ include the L_2-norm

$$\Omega(\theta) = \|\theta\|_2^2 = \sum_{j=1}^{p} \theta_j^2 \tag{15.249}$$

and the L_1-norm

$$\Omega(\theta) = \|\theta\|_1 = \sum_{j=1}^{p} |\theta_j|. \tag{15.250}$$

From a Bayesian perspective, regularization corresponds to imposing a prior distribution $p(\theta)$ on the parameters, such that the posterior distribution $p(\theta|D) \propto p(D|\theta)p(\theta)$ favors simpler models. For L_2 regularization, the prior is a Gaussian distribution

$$p(\theta) \propto \exp\left(-\frac{\lambda}{2}\|\theta\|_2^2\right), \tag{15.251}$$

while for L_1 regularization, the prior is a Laplace distribution

$$p(\theta) \propto \exp(-\lambda\|\theta\|_1). \tag{15.252}$$

Batch normalization (BN) introduces an additional layer of complexity to this framework by normalizing the activations of a neural network within each mini-batch $B = \{x_1, x_2, \ldots, x_m\}$. For a given activation $x \in \mathbb{R}^d$, BN computes the normalized output $\hat{x}$ as:

$$\hat{x} = \frac{x - \mu_B}{\sigma_B^2 + \epsilon}, \tag{15.253}$$

where $\mu_B = \frac{1}{m}\sum_{i=1}^m x_i$ is the mini-batch mean, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^m (x_i - \mu_B)^2$ is the mini-batch variance, and ϵ is a small constant for numerical stability. The normalized output is then scaled and shifted using learnable parameters γ and β, yielding the final output

$$y = \gamma\hat{x} + \beta. \tag{15.254}$$

This transformation ensures that the activations have zero mean and unit variance during training, reducing internal covariate shift and stabilizing the optimization process. The regularization effect of BN arises from its stochastic nature and its impact on the optimization dynamics. During training, the use of mini-batch statistics introduces noise into the gradient updates, which can be modeled as:

$$\tilde{g}(\theta) = g(\theta) + \eta, \tag{15.255}$$

where $g(\theta) = \nabla_\theta L(\theta)$ is the true gradient, $\tilde{g}(\theta)$ is the stochastic gradient computed using BN, and η is a zero-mean random variable with covariance Σ. This noise acts as a form of stochastic regularization, biasing the optimization trajectory toward flatter minima, which are associated with better generalization. The regularization effect can be further analyzed using the continuous-time limit of stochastic gradient descent (SGD), described by the stochastic differential equation (SDE):

$$d\theta_t = -\nabla L_{\mathrm{BN}}(\theta_t)dt + \eta\Sigma dW_t, \tag{15.256}$$

where W_t is a Wiener process. The noise term $\eta\Sigma dW_t$ induces an implicit regularization effect, as it biases the trajectory of θ_t toward regions of the parameter space with smaller curvature. From a theoretical perspective, the regularization effect of BN can be formalized using the PAC-Bayes framework. Let $Q(\theta)$ be a posterior distribution over the parameters induced by BN, and let $P(\theta)$ be a prior distribution. The PAC-Bayes bound states:

$$\mathbb{E}_{\theta\sim Q}[R(\theta)] \leq \mathbb{E}_{\theta\sim Q}[\hat{R}(\theta)] + \mathrm{KL}(Q \parallel P) + \frac{1}{2N}\log\left(\frac{1}{\delta}\right), \tag{15.257}$$

where $\mathrm{KL}(Q \parallel P)$ is the Kullback-Leibler divergence between Q and P. BN reduces $\mathrm{KL}(Q \parallel P)$ by constraining the parameter space, leading to a tighter bound and better generalization. Additionally, BN reduces the effective rank of the activations, leading to a lower-dimensional representation of the data, which further contributes to its regularization effect.

Empirical studies have demonstrated that BN reduces the need for explicit regularization techniques, such as dropout and weight decay, by introducing an implicit regularization effect that is both data-dependent and adaptive. However, the exact form of this implicit regularization remains an open question, and further theoretical analysis is required to fully understand the interaction between BN and other regularization techniques. In conclusion, batch normalization is a powerful tool that not only stabilizes and accelerates training but also introduces a sophisticated form of implicit regularization, which can be rigorously analyzed using tools from statistical learning theory, optimization, and stochastic processes.

15.3.11 Weight Decay

Literature Review: Xu et. al. (2024) [607] introduced a novel dual-phase regularization method that combines excitatory and inhibitory transitions in neural networks. The study highlights the effectiveness of L2 regularization (weight decay) in mitigating overfitting while enhancing convergence speed. This work is critical for researchers looking at biologically inspired regularization techniques. Elshamy et. al. (2024) [608] integrated weight decay regularization into deep learning models for medical imaging. By fine-tuning hyperparameters and regularization techniques, the paper demonstrates improved diagnostic accuracy and robustness against overfitting, making it a crucial reference for medical AI applications. Vinay et. al. (2024) [609] explored L2 regularization (weight decay) and learning rate decay as effective techniques to prevent overfitting in convolutional neural networks (CNNs). It highlights how a structured combination of regularization techniques can improve model robustness in medical image classification. Gai and Huang (2024) [610] introduced a new weight decay method tailored for biquaternion neural networks, emphasizing its role in maintaining balance between model complexity and generalization. It presents rigorous mathematical proofs supporting the effectiveness of weight decay in reducing overfitting. Xu (2025) [611] systematically compared various high-level regularization techniques, including dropout, weight decay, and early stopping, to combat overfitting in deep learning models trained on noisy datasets. It presents empirical evaluations on real-world linkage tasks. Liao et. al. (2025) [612] introduced decay regularization, a variation of weight decay, in stochastic networks to optimize battery Remaining Useful Life (RUL) prediction for UAVs. It provides a novel take on weight decay's impact on sparsification and overfitting control. Dong et. al. (2024) [613] evaluated weight decay in self-knowledge distillation frameworks for improving image classification accuracy. It provides evidence that combining weight decay with knowledge distillation significantly improves model generalization. Ba et. al. (2024) [614] investigated the interplay between data diversity and weight decay regularization in neural networks. The paper introduces a theoretical framework linking weight decay with dataset variability and explores its impact on the weight landscape. Li et. al. (2024) [615] integrated L2 regularization (weight decay) with hybrid data augmentation strategies for audio signal processing, proving its effectiveness in preventing overfitting in deep neural networks. Zang and Yan (2024) [616] presented a new attenuation-based weight decay regularization method for improving network robustness in high-dimensional data scenarios. It introduces novel kernel-learning techniques combined with weight decay for enhanced performance.

Overfitting is a phenomenon that arises when a model $f(x;\theta)$, parameterized by $\theta \in \mathbb{R}^p$, achieves a low empirical risk

$$L_{\text{train}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{15.258}$$

but fails to generalize to unseen data, as quantified by the generalization error

$$L_{\text{test}}(\theta) = \mathbb{E}_{(x,y)\sim P}[\ell(f(x;\theta), y)] \tag{15.259}$$

where P is the true data-generating distribution. The discrepancy between $L_{\text{train}}(\theta)$ and $L_{\text{test}}(\theta)$ is a consequence of the model's excessive capacity to fit noise in the training data, which can be formalized using the Rademacher complexity $R_N(H)$ of the hypothesis space H. Specifically, the Rademacher complexity is defined as

$$R_N(H) = \mathbb{E}_{D,\sigma}\left[\sup_{f\in H}\frac{1}{N}\sum_{i=1}^{N}\sigma_i f(x_i;\theta)\right] \tag{15.260}$$

where σ_i are Rademacher random variables. Overfitting occurs when $R_N(H)$ is large relative to the sample size N, leading to a generalization gap

$$L_{\text{test}}(\theta) - L_{\text{train}}(\theta) \tag{15.261}$$

that grows with the complexity of H. Regularization addresses overfitting by introducing a penalty term $\Omega(\theta)$ into the empirical risk minimization framework, yielding the regularized objective

$$L_{\text{regularized}}(\theta) = L_{\text{train}}(\theta) + \lambda\Omega(\theta) \tag{15.262}$$

where $\lambda > 0$ is the regularization parameter. Weight decay, a specific form of regularization, corresponds to the choice

$$\Omega(\theta) = \frac{1}{2}\|\theta\|_2^2 \tag{15.263}$$

which imposes an L_2 penalty on the model parameters. This penalty can be interpreted as a constraint on the parameter space, restricting the solution to a ball of radius $C = \frac{2}{\lambda}$ in the Euclidean norm, as dictated by the Lagrange multiplier theorem. The regularized objective thus becomes

$$L_{\text{regularized}}(\theta) = L_{\text{train}}(\theta) + \frac{\lambda}{2}\|\theta\|_2^2 \tag{15.264}$$

which is strongly convex if $L_{\text{train}}(\theta)$ is convex, ensuring a unique global minimum θ^*. The optimization dynamics of weight decay can be analyzed through the lens of gradient descent. The update rule for gradient descent with learning rate η and weight decay is given by

$$\theta_{t+1} = \theta_t - \eta\left(\nabla_\theta L_{\text{train}}(\theta_t) + \lambda\theta_t\right) \tag{15.265}$$

which can be rewritten as

$$\theta_{t+1} = (1-\eta\lambda)\theta_t - \eta\nabla_\theta L_{\text{train}}(\theta_t) \tag{15.266}$$

This update rule introduces an exponential decay in the parameter values, ensuring that θ_t remains bounded and converges to the regularized solution θ^*. The convergence properties of this algorithm can be rigorously analyzed using the theory of convex optimization. Specifically, if $L_{\text{train}}(\theta)$ is L-smooth and μ-strongly convex, the regularized objective $L_{\text{regularized}}(\theta)$ is $(L+\lambda)$-smooth and $(\mu+\lambda)$-strongly convex, leading to a linear convergence rate of

$$\|\theta_t - \theta^*\|_2 \leq \left(\frac{L+\lambda}{\mu+\lambda}\right)^t \|\theta_0 - \theta^*\|_2 \tag{15.267}$$

The statistical implications of weight decay can be understood through the bias-variance tradeoff. The bias of the regularized estimator θ^* is given by

$$\text{Bias}(\theta^*) = \mathbb{E}[\theta^*] - \theta_0 \tag{15.268}$$

where θ_0 is the true parameter vector, while the variance is given by

$$\mathrm{Var}(\theta^*) = \mathbb{E}[\|\theta^* - \mathbb{E}[\theta^*]\|_2^2] \tag{15.269}$$

Weight decay increases the bias by shrinking θ^* toward zero but reduces the variance by constraining the parameter space. This tradeoff can be quantified using the ridge regression estimator in the linear model setting, where

$$\theta^* = (X^\top X + \lambda I)^{-1} X^\top y \tag{15.270}$$

The bias and variance of this estimator can be explicitly computed as

$$\mathrm{Bias}(\theta^*) = -\lambda (X^\top X + \lambda I)^{-1} \theta_0 \tag{15.271}$$

and

$$\mathrm{Var}(\theta^*) = \sigma^2 \mathrm{Tr}\left((X^\top X + \lambda I)^{-2} X^\top X\right) \tag{15.272}$$

where σ^2 is the noise variance. The theoretical foundations of weight decay can also be explored through the lens of reproducing kernel Hilbert spaces (RKHS). In this framework, the regularization term $\frac{\lambda}{2}\|\theta\|_2^2$ corresponds to the squared norm in an RKHS H, and the regularized solution is the minimizer of

$$L_{\text{regularized}}(f) = L_{\text{train}}(f) + \frac{\lambda}{2}\|f\|_H^2 \tag{15.273}$$

where $\|f\|_H$ is the norm in H. This connection reveals that weight decay is equivalent to Tikhonov regularization in the RKHS setting, providing a unifying theoretical framework for understanding regularization in both parametric and non-parametric models. In conclusion, weight decay is a mathematically principled regularization technique that addresses overfitting by constraining the hypothesis space and reducing the Rademacher complexity of the model. Its optimization dynamics, statistical properties, and connections to RKHS theory provide a rigorous foundation for understanding its role in improving generalization performance. By carefully tuning the regularization parameter λ, we can achieve an optimal balance between bias and variance, ensuring robust and reliable model performance on unseen data.

15.3.12 Max Norm Constraints

Literature Review: Srivastava et al. (2014) [131] introduced dropout as a regularization method and explores the interplay between dropout and max-norm constraints. The authors show that dropout acts as an implicit regularizer, reducing overfitting by randomly omitting units during training. They also analyze the use of max-norm constraints with dropout, demonstrating that this combination prevents excessive weight growth and stabilizes training in deep neural networks. Moradi et al. (2020) [617] provided a comprehensive survey of regularization techniques, including max-norm constraints. The authors explore different forms of norm-based constraints (L1, L2, and max-norm), discussing their effects on weight magnitude, sparsity, and overfitting reduction. They compare these techniques across multiple neural network architectures. Rodríguez et al. (2016) [618] introduced a novel regularization technique that constrains local weight correlations in CNNs, reducing overfitting without sacrificing learning capacity. They demonstrate that max-norm constraints help prevent weights from growing too large, thus maintaining stability in deep convolutional networks. Tian and Zhang (2022) [619] surveyed different regularization strategies, with a special focus on norm constraints. It extensively discusses the effectiveness of max-norm constraints in preventing overfitting in deep learning models and compares them with weight decay and L1/L2 regularization. Cong et al. (2017) [620] developed a hybrid approach combining max-norm and low-rank constraints to handle overfitting in similarity learning tasks. The authors propose an online learning method that reduces model complexity while maintaining generalization performance. Salman and Liu (2019) [621] conducted an empirical study on how overfitting manifests in deep neural networks and propose max-norm constraints as a key strategy to mitigate overfitting. Their results suggest that max-norm regularization improves generalization by limiting weight magnitudes. Wang et. al. (2021) [622] explored benign overfitting, where models achieve perfect training accuracy but still generalize well. The authors investigate max-norm constraints as a form of implicit regularization and show that they help avoid harmful overfitting in high-dimensional settings. Poggio et al. (2017) [623] presented a theoretical framework explaining why deep networks often avoid overfitting despite having more parameters than data points. They highlight the role of max-norm constraints in controlling model complexity and preventing overfitting. Oyedotun et. al. (2017) [624] discussed the consequences of overfitting in deep networks and compares various norm-based constraints (L1, L2, max-norm). The authors advocate for max-norm regularization due to its computational efficiency and robustness in high-dimensional spaces. Luo et al. (2016) [625] proposed an improved extreme learning machine (ELM) model that integrates L1, L2, and max-norm constraints to enhance generalization performance. The authors show that max-norm regularization effectively prevents overfitting while maintaining model interpretability.

Overfitting is a fundamental problem in machine learning that occurs when a model captures noise or spurious patterns in the training data instead of learning the underlying distribution. Mathematically, overfitting can be understood in terms of generalization error, which is the discrepancy between the empirical risk $\mathcal{L}_{\text{empirical}}(\mathbf{w})$ and the expected risk $\mathcal{L}(\mathbf{w})$. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the model is parameterized by $\mathbf{w}$ and optimized to minimize the empirical risk

$$\mathcal{L}_{\text{empirical}}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\mathbf{w}}(\mathbf{x}_i), y_i) \tag{15.274}$$

where $\ell(\cdot,\cdot)$ is a loss function, such as the squared loss for regression:

$$\ell(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \frac{1}{2}\left(f_{\mathbf{w}}(\mathbf{x}_i) - y_i\right)^2 \tag{15.275}$$

However, the expected risk, which measures the model's true generalization performance on unseen data, is given by

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(f_{\mathbf{w}}(\mathbf{x}), y)] \tag{15.276}$$

The generalization gap is defined as

$$\mathcal{L}(\mathbf{w}) - \mathcal{L}_{\text{empirical}}(\mathbf{w}) \tag{15.277}$$

and it increases when the model complexity is too high relative to the number of training samples. In statistical learning theory, this gap can be upper-bounded using the Vapnik-Chervonenkis (VC) dimension $\mathsf{VC}(\mathcal{H})$ of the hypothesis class $\mathcal{H}$, yielding the bound

$$\mathbb{E}[\mathcal{L}(\mathbf{w})] \leq \mathcal{L}_{\text{empirical}}(\mathbf{w}) + O\left(\sqrt{\frac{\mathsf{VC}(\mathcal{H})}{N}}\right) \tag{15.278}$$

This inequality suggests that models with high VC dimension have larger generalization gaps, leading to overfitting. Another theoretical measure of complexity is the Rademacher complexity, which quantifies the ability of a function class to fit random noise. If $\mathcal{H}$ has high Rademacher complexity $\mathcal{R}(\mathcal{H})$, the generalization bound

$$\mathbb{E}[\mathcal{L}(\mathbf{w})] \leq \mathcal{L}_{\text{empirical}}(\mathbf{w}) + O\left(\mathcal{R}(\mathcal{H})\right) \tag{15.279}$$

indicates poor generalization. Regularization techniques aim to reduce the effective hypothesis space, thereby improving generalization by controlling model complexity. One effective approach to mitigating overfitting is the incorporation of a regularization term in the objective function. A general regularized loss function takes the form

$$\mathcal{L}_\lambda(\mathbf{w}) = \mathcal{L}_{\text{empirical}}(\mathbf{w}) + \lambda\Omega(\mathbf{w}) \tag{15.280}$$

where $\Omega(\mathbf{w})$ is a penalty function enforcing constraints on $\mathbf{w}$, and λ is a hyperparameter controlling the strength of regularization. Popular choices for $\Omega(\mathbf{w})$ include the L_2 norm (ridge regression)

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2 \tag{15.281}$$

which shrinks large weight values but does not impose an explicit bound on their magnitude. Similarly, L_1 regularization (lasso regression),

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j| \tag{15.282}$$

promotes sparsity but does not constrain the overall norm. Max-norm regularization is a stricter form of regularization that directly enforces an upper bound on the norm of the weight vector. Specifically, it constrains the weight norm to satisfy

$$\|\mathbf{w}\|_2 \leq c \tag{15.283}$$

for some constant c. This constraint prevents the optimizer from selecting solutions where the weight magnitudes grow excessively, thereby controlling model complexity more effectively than L_2 regularization. Instead of adding a penalty term to the loss function, max-norm regularization enforces the constraint during optimization by projecting the weight vector onto the feasible set whenever it exceeds the bound. Mathematically, this projection step is given by

$$\mathbf{w} \leftarrow \frac{c}{\max(\|\mathbf{w}\|_2, c)}\mathbf{w} \tag{15.284}$$

From a geometric perspective, max-norm regularization restricts the hypothesis space to a Euclidean ball of radius c centered at the origin. The restricted hypothesis space leads to a lower VC dimension and reduced Rademacher complexity, improving generalization. The constrained optimization problem can be reformulated using Lagrange multipliers, leading to the constrained optimization problem

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \text{subject to} \quad \|\mathbf{w}\|_2 \leq c \tag{15.285}$$

Introducing the Lagrange multiplier α, the Lagrangian function is

$$\mathcal{L}_\alpha(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \alpha(\|\mathbf{w}\|_2^2 - c^2) \tag{15.286}$$

Differentiating with respect to $\mathbf{w}$ gives the optimality condition

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) + 2\alpha\mathbf{w} = 0 \tag{15.287}$$

Solving for $\mathbf{w}$, we obtain

$$\mathbf{w} = -\frac{1}{2\alpha}\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) \tag{15.288}$$

which shows that weight updates are constrained in a direction dependent on α, effectively controlling their magnitude.

15.3.13 Transfer Learning

Literature Review: Cakmakci [598] examined the use of Xception-based transfer learning in pediatric bone age prediction. It highlights the importance of dropout regularization in preventing overfitting in deep models trained on small datasets. The paper provides insights into how regularization techniques can maintain model generalizability. Zhou et. al. (2024) [626] focused on ElasticNet regularization combined with transfer learning to prevent overfitting in rice disease classification. The research demonstrates that L1 and L2 regularization can significantly improve generalization by penalizing model complexity, especially in scenarios with limited labeled data. Omole et. al. (2024) [627] explored Neural Architecture Search (NAS) with transfer learning, integrating adaptive convolution and regularization-based techniques to enhance model robustness. The authors implement batch normalization and weight decay to address overfitting issues common in agricultural image datasets. By leveraging data augmentation, dropout, and fine-tuning, Tripathi, et. al. (2024) [628] optimized a VGG-16-based transfer learning approach for brain tumor detection. The study shows how dropout regularization and L2 penalty mitigate overfitting and improve model robustness when handling medical images. Singla and Gupta [629] emphasized early stopping, dropout regularization, and L1/L2 penalties in preventing overfitting in transfer learning models applied to medical imaging. The authors highlight the impact of model complexity on overfitting and suggest hyperparameter tuning as a complementary solution. Adhaileh et. al. (2024) [630] introduced a multi-phase transfer learning model with regularization-based fine-tuning to enhance diagnostic accuracy in chest disease classification. The study integrates batch normalization, weight decay, and dropout layers to prevent overfitting in CNN-based architectures. Harvey et. al. (2025) [631] presented a data-driven hyperparameter optimization technique that adapts regularization strength dynamically. The proposed L2-zero regularization method adjusts the weight penalty based on the importance of data samples, improving transfer learning model robustness against overfitting. Mahmood et. al. (2025) [632] introduced regional regularization loss functions in transfer learning for medical imaging. It focuses on mitigating overfitting through adversarial training and data augmentation, ensuring robustness across diverse datasets. Shen (2025) [633] combined feature selection with transfer learning to prevent overfitting in sports analytics. The study highlights Ridge and Lasso regularization as essential tools in stabilizing model predictions in high-dimensional data. Guo et. al. (2025) [634] developed uncertainty-aware knowledge distillation for transfer learning in medical image segmentation. It employs cyclic ensemble training and dropout-based uncertainty estimation to mitigate overfitting and improve generalization performance.

Let's discuss the Mathematical Formulation of Transfer Learning and Overfitting. Let $\mathcal{X} \subset \mathbb{R}^d$ be the input space and $\mathcal{Y}$ be the label space. In **transfer learning**, we assume the existence of two probability distributions: the **source distribution** $\mathcal{P}_{\text{source}}(\mathbf{x}, y)$ and the **target distribution** $\mathcal{P}_{\text{target}}(\mathbf{x}, y)$, which govern the input-output relationship. The goal of transfer learning is to approximate the **optimal target hypothesis function** $f^*(\mathbf{x})$ by leveraging knowledge from the source model $f_s(\mathbf{x})$, while minimizing the **expected risk** over the target distribution:

$$\mathcal{R}_{\text{target}}(f) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{\text{target}}} \left[\mathcal{L}(f(\mathbf{x}), y)\right]. \tag{15.289}$$

Since $\mathcal{P}_{\text{target}}$ is unknown, we approximate $\mathcal{R}_{\text{target}}(f)$ using the **empirical risk** computed over a finite dataset $\mathcal{D}_{\text{target}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$:

$$\hat{\mathcal{R}}_{\text{target}}(f) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i), y_i). \tag{15.290}$$

A model that **perfectly minimizes** $\hat{\mathcal{R}}_{\text{target}}(f)$ may lead to **overfitting**, wherein the function $f(\mathbf{x})$ aligns with noise in the training set instead of generalizing well to new data. The degree of overfitting is measured by the **generalization gap**:

$$\mathcal{G}(f) = \mathcal{R}_{\text{target}}(f) - \hat{\mathcal{R}}_{\text{target}}(f). \tag{15.291}$$

According to **Statistical Learning Theory**, the **generalization error bound** is governed by the **Rademacher complexity** $\mathfrak{R}(\mathcal{H})$ of the hypothesis space $\mathcal{H}$, which quantifies the capacity of $\mathcal{H}$ to fit random noise:

$$\mathcal{G}(f) \leq \mathcal{O}\left(\mathfrak{R}(\mathcal{H}) + \sqrt{\frac{\log N}{N}}\right). \tag{15.292}$$

This implies that hypothesis spaces with **high Rademacher complexity** suffer from large generalization gaps, leading to overfitting. Regularization can be thought as a Mechanism for Controlling Hypothesis Complexity. To mitigate overfitting, we impose a **regularization functional** $\Omega(f)$ that penalizes excessively complex hypotheses. This modifies the optimization problem to:

$$f^* = \arg\min_{f\in\mathcal{H}} \left[\hat{\mathcal{R}}_{\text{target}}(f) + \lambda\Omega(f)\right]. \tag{15.293}$$

where λ is a **hyperparameter** balancing empirical risk minimization and model complexity. From the perspective of **functional analysis**, we interpret regularization as imposing constraints on the **function space** where f is chosen. In many cases, f is assumed to belong to a **Reproducing Kernel Hilbert Space (RKHS)** $\mathcal{H}_K$ associated with a kernel function $K(\mathbf{x}, \mathbf{x}')$. The RKHS norm,

$$\|f\|^2_{\mathcal{H}_K} = \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{15.294}$$

acts as a smoothness regularizer that prevents excessive function oscillations. Alternatively, in the **Sobolev space** $W^{m,p}(\mathcal{X})$, regularization can take the form:

$$\Omega(f) = \int_{\mathcal{X}} |D^m f(\mathbf{x})|^p \, d\mathbf{x}, \tag{15.295}$$

where $D^m f$ represents the mth weak derivative of f. The choice of m and p dictates the smoothness constraints imposed on f, directly influencing its generalization ability. One of the most widely used regularization techniques is **L2 regularization** or **Tikhonov regularization**, which penalizes the Euclidean norm of the model parameters:

$$\Omega(f) = \|\theta\|_2^2 = \sum_i \theta_i^2. \tag{15.296}$$

To understand the effect of L2 regularization, consider the **Hessian matrix** $H = \nabla_\theta^2 \mathcal{L}$, which captures the local curvature of the loss landscape. The largest eigenvalue $\lambda_{\max}$ determines the sharpness of the loss minimum:

$$\|H\|_2 = \sup_{\|\mathbf{v}\|_2=1} \|H\mathbf{v}\|_2. \tag{15.297}$$

A sharp minimum, corresponding to a high $\lambda_{\max}$, leads to poor generalization. L2 regularization modifies the eigenvalue spectrum of the Hessian, effectively reducing $\lambda_{\max}$, leading to smoother loss surfaces and improved generalization. In conclusion, The Bias-Variance Tradeoff and Optimal Regularization Selection, Regularization directly influences the **bias-variance tradeoff**:

- **Under-regularization**: Low bias, high variance ⇒ overfitting.
- **Over-regularization**: High bias, low variance ⇒ underfitting.

By tuning λ via **cross-validation**, we achieve a balance between **empirical risk minimization** and **hypothesis complexity control**, ensuring **optimal generalization performance**.

15.4 Hyperparameter Tuning

Literature Review: Luo et. al. (2003) [137] provided a deep dive into Bayesian Optimization, a widely used method for hyperparameter tuning. It covers theoretical foundations, practical applications, and advanced strategies for establishing an appropriate range for hyperparameters. This resource is essential for researchers interested in probabilistic approaches to tuning machine learning models. Alrayes et. al. (2025) [138] explored the use of statistical learning and optimization algorithms to fine-tune hyperparameters in machine learning models applied to IoT networks. The paper emphasizes privacy-preserving approaches, making it valuable for practitioners working with secure data environments. Cho et. al. (2020) [139] discussed basic enhancement strategies when using Bayesian Optimization for Hyperparameter Tuning of Deep Neural Networks. Ibrahim et. al. (2025) [140] focused on hyperparameter tuning for XGBoost, a widely used machine learning model, in the context of medical diagnosis. It showcases a comparative analysis of tuning techniques to optimize model performance in real-world healthcare applications. Abdel-Salam et. al. (2025) [141] introduced an evolved framework for tuning deep learning models using multiple optimization algorithms. It presented a novel approach that outperforms traditional techniques in training deep networks. Vali (2025) [142] in his Doctoral thesis covers how vector quantization techniques aid in reducing hyperparameter search space for deep learning models. It emphasizes computational efficiency in speech and image processing applications. Vincent and Jidesh (2023) [143] in their paper explored various hyperparameter optimization techniques, comparing their performance on image classification datasets using AutoML models. It focuses on Bayesian optimization and introduces genetic algorithms, differential evolution, and covariance matrix adaptation—evolutionary strategy (CMA-ES) for acquisition function optimization. Results show that CMA-ES and differential evolution enhance Bayesian optimization, while genetic algorithms degrade its performance. Razavi-Termeh et. al. (2025) [144] explored the role of geospatial artificial intelligence (GeoAI) in mapping flood-prone areas, leveraging metaheuristic algorithms for hyperparameter tuning. It offers insights into machine learning applications in environmental science. Kiran and Ozyildirim (2022) [145] proposed a distributed variable-length genetic algorithm to optimize hyperparameters in reinforcement learning (RL), improving training efficiency and robustness. Unlike traditional deep RL, which lacks extensive tuning due to complexity, our approach systematically enhances performance across various RL tasks, outperforming Bayesian methods. Results show that more generations yield optimal, computationally efficient solutions, advancing RL for real-world applications.

Hyperparameter tuning in neural networks represents an intricate, highly mathematical optimization challenge that is fundamental to achieving optimal performance on a given task. This process can be framed as a bi-level optimization problem, where the outer optimization concerns the selection of hyperparameters $h \in \mathcal{H}$ to minimize a validation loss function $\mathcal{L}_{\text{val}}(\theta^*(h); h)$, while the inner optimization determines the optimal model parameters θ^* by minimizing the training loss $\mathcal{L}_{\text{train}}(\theta; h)$. This can be expressed rigorously as follows:

$$h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}_{\text{val}}(\theta^*(h); h), \quad \text{where} \quad \theta^*(h) = \arg\min_{\theta} \mathcal{L}_{\text{train}}(\theta; h). \tag{15.298}$$

Here, $\mathcal{H}$ denotes the hyperparameter space, which is often high-dimensional, non-convex, and computationally expensive to traverse. The training loss function $\mathcal{L}_{\text{train}}(\theta; h)$ is typically represented as an empirical risk computed over the training dataset $\{(x_i, y_i)\}_{i=1}^N$:

$$\mathcal{L}_{\text{train}}(\theta; h) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta, h), y_i), \tag{15.299}$$

where $f(x_i; \theta, h)$ is the neural network output given the input x_i, parameters θ, and hyperparameters h, and $\ell(a, b)$ is the loss function quantifying the discrepancy between prediction a and ground truth

b. For classification tasks, ℓ often takes the form of cross-entropy loss:

$$\ell(a, b) = -\sum_{k=1}^{C} b_k \log a_k, \tag{15.300}$$

where C is the number of classes, and a_k and b_k are the predicted and true probabilities for the k-th class, respectively. Central to the training process is the optimization of θ via gradient-based methods such as stochastic gradient descent (SGD). The parameter updates are governed by:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(t)}; h), \tag{15.301}$$

where $\eta > 0$ is the learning rate, a critical hyperparameter controlling the step size. The stability and convergence of SGD depend on η, which must satisfy:

$$0 < \eta < \frac{2}{\lambda_{\max}(H)}, \tag{15.302}$$

where $\lambda_{\max}(H)$ is the largest eigenvalue of the Hessian matrix $H = \nabla_\theta^2 \mathcal{L}_{\text{train}}(\theta; h)$. This condition ensures that the gradient descent steps do not overshoot the minimum. To analyze convergence behavior, the loss function $\mathcal{L}_{\text{train}}(\theta; h)$ near a critical point θ^* can be approximated via a second-order Taylor expansion:

$$\mathcal{L}_{\text{train}}(\theta; h) \approx \mathcal{L}_{\text{train}}(\theta^*; h) + \frac{1}{2}(\theta - \theta^*)^\top H (\theta - \theta^*), \tag{15.303}$$

where H is the Hessian matrix of second derivatives. The eigenvalues of H reveal the local curvature of the loss surface, with positive eigenvalues indicating directions of convexity and negative eigenvalues corresponding to saddle points. Regularization is often introduced to improve generalization by penalizing large parameter values. For L_2 regularization, the modified training loss is:

$$\mathcal{L}_{\text{train}}^{\text{reg}}(\theta; h) = \mathcal{L}_{\text{train}}(\theta; h) + \frac{\lambda}{2}\|\theta\|_2^2, \tag{15.304}$$

where $\lambda > 0$ is the regularization coefficient. The gradient of the regularized loss becomes:

$$\nabla_\theta \mathcal{L}_{\text{train}}^{\text{reg}}(\theta; h) = \nabla_\theta \mathcal{L}_{\text{train}}(\theta; h) + \lambda\theta. \tag{15.305}$$

Another key hyperparameter is the weight initialization strategy, which affects the scale of activations and gradients throughout the network. For a layer with n_{in} inputs, He initialization samples weights from:

$$w_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right), \tag{15.306}$$

to ensure that the variance of activations remains stable as data propagate through layers. The activation function $g(z)$ also plays a crucial role. The Rectified Linear Unit (ReLU), defined as $g(z) = \max(0, z)$, introduces sparsity and mitigates vanishing gradients. However, it suffers from the "dying neuron" problem, as its derivative $g'(z)$ is zero for $z \leq 0$. The search for optimal hyperparameters can be approached using grid search, random search, or more advanced methods like Bayesian optimization. In Bayesian optimization, a surrogate model $p(\mathcal{L}_{\text{val}}(h))$, often a Gaussian Process (GP), is constructed to approximate the validation loss. The acquisition function $a(h)$, such as Expected Improvement (EI), guides the exploration of $\mathcal{H}$ by balancing exploitation of regions with low predicted loss and exploration of uncertain regions:

$$a(h) = \mathbb{E}\left[\max(0, \mathcal{L}_{\text{val,min}} - \mathcal{L}_{\text{val}}(h))\right], \tag{15.307}$$

where $\mathcal{L}_{\text{val,min}}$ is the best observed validation loss. Hyperparameter tuning is computationally intensive due to the high dimensionality of $\mathcal{H}$ and the nested nature of the optimization problem.

Early stopping, a widely used strategy, halts training when the improvement in validation loss falls below a threshold:

$$\frac{\mathcal{L}_{\text{val}}^{(t+1)} - \mathcal{L}_{\text{val}}^{(t)}}{\mathcal{L}_{\text{val}}^{(t)}} < \epsilon, \tag{15.308}$$

where $\epsilon > 0$ is a small constant. Advanced techniques like Hyperband leverage multi-fidelity optimization, allocating resources dynamically to promising hyperparameter configurations based on partial training evaluations.

In conclusion, hyperparameter tuning for training neural networks is an exceptionally mathematically rigorous process, grounded in nested optimization, gradient-based methods, probabilistic modeling, and computational heuristics. Each component, from learning rates and regularization to initialization and optimization strategies, contributes to the complex interplay that defines neural network performance.

15.4.1 Grid Search

Literature Review: Rohman and Farikhin (2025) [397] explored the impact of Grid Search and Random Search in hyperparameter tuning for Random Forest classifiers in the context of diabetes prediction. The study provides a comparative analysis of different hyperparameter tuning strategies and demonstrates that Grid Search improves classification accuracy by selecting optimal hyperparameter combinations systematically. Rohman (2025) [398] applied Grid Search-based hyperparameter tuning to optimize machine learning models for early brain tumor detection. The study emphasizes the importance of systematic hyperparameter selection and provides insights into how Grid Search affects diagnostic accuracy and computational efficiency in medical applications. Nandi et al. (2025) [399] examined the use of Grid Search for deep learning hyperparameter tuning in baby cry sound recognition systems. The authors present a novel pipeline that systematically selects the best hyperparameters for neural networks, improving both precision and recall in sound classification. Sianga et. al. (2025) [400] applied Grid Search and Randomized Search to optimize machine learning models predicting cardiovascular disease risk. The study finds that Grid Search consistently outperforms randomized methods in accuracy, highlighting its effectiveness in medical diagnostic models. Li et. al. (2025) [401] applied Stratified 5-fold cross-validation combined with Grid Search to fine-tune Extreme Gradient Boosting (XGBoost) models in predicting post-surgical complications. The results suggest that hyperparameter tuning significantly improves predictive performance, with Grid Search leading to the best model stability and interpretability. Lázaro et. al. (2025) [402] implemented Grid Search and Bayesian Optimization to optimize K-Nearest Neighbors (KNN) and Decision Trees for incident classification in aviation safety. The research underscores how different hyperparameter tuning methods affect the generalization of machine learning models in NLP-based accident reports. Li et. al. (2025) [403] proposed RAINER, an ensemble learning model that integrates Grid Search for optimal hyperparameter tuning. The study demonstrates how parameter optimization enhances the predictive capabilities of rainfall models, making Grid Search an essential step in climate modeling. Khurshid et. al. (2025) [404] compared Bayesian Optimization with Grid Search for hyperparameter tuning in diabetes prediction models. The study finds that while Bayesian methods are computationally faster, Grid Search delivers more precise hyperparameter selection, especially for models with structured medical data. Kanwar et. al. (2025) [405] applied Grid Search for tuning Random Forest classifiers in landslide susceptibility mapping. The study demonstrates that fine-tuned models improve the identification of high-risk zones, reducing false positives in predictive landslide models. Fadil et. al. (2025) [406] evaluated the role of Grid Search and Random Search in hyperparameter tuning for XGBoost regression models in corrosion prediction. The authors find that Grid Search-based models achieve higher R^2 scores, making them ideal for complex chemical modeling applications.

Grid search is a highly structured and exhaustive method for hyperparameter tuning in machine learning, where a predetermined grid of hyperparameter values is systematically explored. The goal is to identify the set of hyperparameters $\vec{h} = (h_1, h_2, \ldots, h_p)$ that yields the optimal performance metric for a given machine learning model. Let p represent the total number of hyperparameters to be tuned, and for each hyperparameter h_i, let the candidate set be $\mathcal{H}_i = \{h_{i1}, h_{i2}, \ldots, h_{im_i}\}$, where m_i is the number of candidate values for h_i. The hyperparameter search space is then the Cartesian product of all candidate sets:

$$\mathcal{S} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_p. \tag{15.309}$$

Thus, the total number of configurations to be evaluated is:

$$|\mathcal{S}| = \prod_{i=1}^{p} m_i. \tag{15.310}$$

For example, if we have two hyperparameters h_1 and h_2 with 3 possible values each, the total number of combinations to explore is 9. This search space grows exponentially as the number of hyperparameters increases, posing a significant computational challenge. Grid search involves iterating over all configurations in $\mathcal{S}$, evaluating the model's performance for each configuration.

Let us define the performance metric $M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}})$, which quantifies the model's performance for a given hyperparameter configuration $\vec{h}$, where $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ are the training and validation datasets, respectively. This metric might represent accuracy, error rate, F1-score, or any other relevant criterion, depending on the problem at hand. The hyperparameters are then tuned by maximizing or minimizing M across the search space:

$$\vec{h}^* = \arg\max_{\vec{h}\in\mathcal{S}} M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}), \tag{15.311}$$

or in the case of a minimization problem:

$$\vec{h}^* = \arg\min_{\vec{h}\in\mathcal{S}} M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}). \tag{15.312}$$

For each hyperparameter combination, the model is trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$. The process requires the repeated evaluation of the model over all $|\mathcal{S}|$ configurations, each yielding a performance metric. To mitigate overfitting and ensure the reliability of the performance metric, cross-validation is frequently used. In k-fold cross-validation, the dataset $\mathcal{D}_{\text{train}}$ is partitioned into k disjoint subsets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$. The model is trained on $\mathcal{D}_{\text{train}}^{(j)} = \bigcup_{i\neq j} \mathcal{D}_i$ and validated on $\mathcal{D}_j$. For each fold j, we compute the performance metric:

$$M_j(\vec{h}) = M(\vec{h}, \mathcal{D}_{\text{train}}^{(j)}, \mathcal{D}_j). \tag{15.313}$$

The overall cross-validation performance for a hyperparameter configuration $\vec{h}$ is the average of the k individual fold performances:

$$\overline{M}(\vec{h}) = \frac{1}{k}\sum_{j=1}^{k} M_j(\vec{h}). \tag{15.314}$$

Thus, the grid search with cross-validation aims to find the optimal hyperparameters by maximizing or minimizing the average performance across all folds. The computational complexity of grid search is a key consideration. If we denote C as the cost of training and evaluating the model for a single configuration, the total cost for grid search is:

$$\mathcal{O}\left(\prod_{i=1}^{p} m_i \cdot k \cdot C\right), \tag{15.315}$$

where k represents the number of folds in cross-validation. This results in an exponential increase in the total computation time as the number of hyperparameters p and the number of candidate values m_i increase. For large search spaces, grid search can become computationally expensive, making it infeasible for high-dimensional hyperparameter optimization problems. To illustrate with a specific example, consider two hyperparameters h_1 and h_2 with the following sets of candidate values:

$$\mathcal{H}_1 = \{0.01, 0.1, 1.0\}, \quad \mathcal{H}_2 = \{0.1, 1.0, 10.0\}. \tag{15.316}$$

The search space is:

$$\mathcal{S} = \mathcal{H}_1 \times \mathcal{H}_2 = \{(0.01, 0.1), (0.01, 1.0), (0.01, 10.0), (0.1, 0.1), \dots, (1.0, 10.0)\}. \tag{15.317}$$

There are 9 configurations to evaluate. For each configuration, assume we perform 3-fold cross-validation, where the performance metrics for the first fold are:

$$M_1(0.1, 1.0) = 0.85, \quad M_2(0.1, 1.0) = 0.87, \quad M_3(0.1, 1.0) = 0.86, \tag{15.318}$$

giving the cross-validation performance:

$$\overline{M}(0.1, 1.0) = \frac{1}{3}\sum_{j=1}^{3} M_j(0.1, 1.0) = \frac{1}{3}(0.85 + 0.87 + 0.86) = 0.86. \tag{15.319}$$

This process is repeated for all 9 combinations of h_1 and h_2. Grid search, while exhaustive and deterministic, can fail to efficiently explore the hyperparameter space, especially when the number of hyperparameters is large. The search is confined to a discrete grid and cannot interpolate between points to capture optimal configurations that may lie between grid values. Furthermore, because grid search evaluates each configuration independently, it can be computationally expensive for high-dimensional spaces, as the number of configurations grows exponentially with p and m_i.

In conclusion, grid search is a methodologically rigorous and systematic approach to hyperparameter optimization, ensuring that all predefined configurations are evaluated exhaustively. However, its computational cost increases exponentially with the number of hyperparameters and their respective candidate values, which can limit its applicability for large-scale problems. As a result, more advanced techniques such as random search, Bayesian optimization, or evolutionary algorithms are often used for hyperparameter tuning when the computational budget is limited. Despite these challenges, grid search remains a powerful tool for demonstrating the principles of hyperparameter tuning and is well-suited for problems with relatively small search spaces. The pros of grid search are:

1. Guaranteed to find the best combination within the search space.

2. Easy to implement and parallelize.

The cons of grid search are:

1. Computationally expensive, especially for high-dimensional hyperparameter spaces.

2. Inefficient if some hyperparameters have little impact on performance.

15.4.2 Random Search

Literature Review: Sianga et. al. (2025) [400] explored Random Search vs. Grid Search for tuning machine learning models in cardiovascular disease risk prediction. It finds that Random Search significantly reduces computation time while maintaining high accuracy, making it preferable for high-dimensional datasets in medical applications. Lázaro et. al. (2025) [402] applied

Random Search and Grid Search to optimize models for accident classification using NLP. The study highlights Random Search's efficiency in tuning K-Nearest Neighbors (KNN) and Decision Trees, leading to faster convergence with minimal loss in accuracy. Emmanuel et. al. (2025) [407] introduced a hybrid approach combining Random Search with Differential Evolution optimization to enhance deep-learning-based protein interaction models. The study demonstrates how Random Search improves generalization and reduces overfitting. Gaurav et. al. (2025) [408] evaluated Random Search optimization in Random Forest classifiers for driver identification. They compare Random Search, Bayesian Optimization, and Genetic Algorithms, concluding that Random Search provides a balance between efficiency and performance. Kanwar et. al. (2025) [405] applied Random Search hyperparameter tuning to Random Forest models for landslide risk assessment. It finds that Random Search significantly reduces computation time without compromising model accuracy, making it ideal for large-scale geospatial analyses. Ning et al. (2025) [409] evaluated Random Search for optimizing mortality prediction models in infected pancreatic necrosis patients. The authors conclude that Random Search outperforms exhaustive Grid Search in finding optimal hyperparameters with significant speed improvements. Muñoz et. al. (2025) [410] presented a novel optimization strategy that combines Random Search with a secretary algorithm to improve hyperparameter tuning efficiency. It demonstrates how Random Search can be adapted to dynamic optimization problems in real-time AI applications. Balcan et. al. (2025) [411] explored the theoretical underpinnings of Random Search in deep learning optimization. They provide a rigorous analysis of the sample complexity required for effective tuning, establishing mathematical guarantees for Random Search efficiency. Azimi et. al. (2025) [412] compared Random Search with metaheuristic algorithms (e.g., Genetic Algorithms and Particle Swarm Optimization) in supercapacitor modeling. The results indicate that Random Search provides a robust baseline for hyperparameter optimization in deep learning models. Shibina and Thasleema (2025) [413] applied Random Search for optimizing ensemble learning classifiers in medical diagnosis. The results show Random Search's advantage in finding optimal hyperparameters for detecting Parkinson's disease using voice features, making it a practical alternative to Bayesian Optimization.

In machine learning, hyperparameter tuning is the process of selecting the best configuration of hyperparameters $\mathbf{h} = (h_1, h_2, \ldots, h_d)$, where each h_i represents the i-th hyperparameter. The hyperparameters $\mathbf{h}$ control key aspects of model learning, such as the learning rate, regularization strength, or the architecture of the neural network. These hyperparameters are not directly optimized through the learning process itself but are instead set before training begins. Given a set of hyperparameters, the model performance is evaluated by computing a loss function $L(\mathbf{h})$, which typically represents the error on a validation set, and possibly regularization terms to mitigate overfitting. The objective is to minimize this loss function to find the optimal set of hyperparameters:

$$\mathbf{h}^* = \arg\min_{\mathbf{h}} L(\mathbf{h}), \tag{15.320}$$

where $L(\mathbf{h})$ is the loss function that quantifies how well the model generalizes to unseen data. The minimization of this function is often subject to constraints on the range or type of values that each h_i can take, forming a constrained optimization problem:

$$\mathbf{h}^* = \arg\min_{\mathbf{h}\in\mathcal{H}} L(\mathbf{h}), \tag{15.321}$$

where $\mathcal{H}$ represents the feasible hyperparameter space. Hyperparameter tuning is typically carried out by selecting a search method that explores this space efficiently, with the goal of finding the global or local optimum of the loss function.

One such search method is **random search**, which is a straightforward yet effective approach to exploring the hyperparameter space. Instead of exhaustively searching over a grid of values for each hyperparameter (as in grid search), random search samples hyperparameters $\mathbf{h}_t =$

$(h_{t,1}, h_{t,2}, \dots, h_{t,d})$ from a predefined distribution for each hyperparameter h_i. For each iteration t, the hyperparameters are independently sampled from probability distributions $\mathcal{D}_i$ associated with each hyperparameter h_i, where the probability distribution might be continuous or discrete. Specifically, for continuous hyperparameters, $h_{t,i}$ is drawn from a uniform or normal distribution over an interval $H_i = [a_i, b_i]$:

$$h_{t,i} \sim \mathcal{U}(a_i, b_i), \quad h_{t,i} \in H_i, \tag{15.322}$$

where $\mathcal{U}(a_i, b_i)$ denotes the uniform distribution between a_i and b_i. For discrete hyperparameters, $h_{t,i}$ is sampled from a discrete set of values $H_i = \{h_{i1}, h_{i2}, \dots, h_{iN_i}\}$ with each value equally probable:

$$h_{t,i} \sim \mathcal{D}_i, \quad h_{t,i} \in \{h_{i1}, h_{i2}, \dots, h_{iN_i}\}, \tag{15.323}$$

where $\mathcal{D}_i$ denotes the discrete distribution over the set $\{h_{i1}, h_{i2}, \dots, h_{iN_i}\}$. Thus, each hyperparameter is selected independently from its corresponding distribution. After selecting a new set of hyperparameters $\mathbf{h}_t$, the model is trained with this configuration, and its performance is evaluated by computing the loss function $L(\mathbf{h}_t)$. The process is repeated for T iterations, generating a sequence of hyperparameter configurations $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$, and for each configuration, the associated loss function values $L(\mathbf{h}_1), L(\mathbf{h}_2), \dots, L(\mathbf{h}_T)$ are computed. The optimal set of hyperparameters $\mathbf{h}^*$ is then selected as the one that minimizes the loss:

$$\mathbf{h}^* = \arg \min_{t \in \{1,2,\dots,T\}} L(\mathbf{h}_t). \tag{15.324}$$

Thus, random search performs an approximate optimization of the hyperparameter space, where the computational cost per iteration is C (the time to evaluate the model's performance for a given set of hyperparameters), and the total computational cost is $O(T \cdot C)$. This makes random search a computationally feasible approach, especially when T is moderate. The computational efficiency of random search can be compared to that of grid search, which exhaustively searches the hyperparameter space by discretizing each hyperparameter h_i into a set of values $h_{i1}, h_{i2}, \dots, h_{in_i}$, where n_i is the number of values for the i-th hyperparameter. The total number of grid search configurations is given by:

$$N_{\text{grid}} = \prod_{i=1}^{d} n_i, \tag{15.325}$$

and the computational cost of grid search is $O(N_{\text{grid}} \cdot C)$, which grows exponentially with the number of hyperparameters d. In this sense, grid search can become prohibitively expensive when the dimensionality d of the hyperparameter space is large. Random search, on the other hand, requires only T evaluations, and since each evaluation is independent of the others, the computational cost grows linearly with T, making it more efficient when d is large. The probabilistic nature of random search further enhances its efficiency. Suppose that only a subset of hyperparameters, say k, significantly influences the model's performance. Let S be the subspace of $\mathcal{H}$ consisting of hyperparameter configurations that produce low loss values, and let the complementary space $\mathcal{H} \backslash S$ correspond to configurations that are unlikely to achieve low loss. In this case, the task becomes one of searching within the subspace S, rather than the entire space $\mathcal{H}$. The random search method is well-suited to such problems, as it can probabilistically focus on the relevant subspace by drawing hyperparameter values from distributions $\mathcal{D}_i$ that prioritize areas of the hyperparameter space with low loss. More formally, the probability of selecting a hyperparameter set $\mathbf{h}_t$ from the relevant subspace S is given by:

$$P(\mathbf{h}_t \in S) = \prod_{i=1}^{d} P(h_{t,i} \in S_i), \tag{15.326}$$

where S_i is the relevant region for the i-th hyperparameter, and $P(h_{t,i} \in S_i)$ is the probability that the i-th hyperparameter lies within the relevant region. As the number of iterations T increases, the probability that random search selects a hyperparameter set $\mathbf{h}_t \in S$ increases as well, approaching 1 as $T \to \infty$:

$$P(\mathbf{h}_t \in S) = 1 - (1 - P_0)^T, \tag{15.327}$$

where P_0 is the probability of sampling a hyperparameter set from the relevant subspace in one iteration. Thus, random search tends to explore the subspace of low-loss configurations, improving the chances of finding an optimal or near-optimal configuration as T increases.

The exploration behavior of random search contrasts with that of grid search, which, despite its systematic nature, may fail to efficiently explore sparsely populated regions of the hyperparameter space. When the hyperparameter space is high-dimensional, the grid search must evaluate exponentially many configurations, regardless of the relevance of the hyperparameters. This leads to inefficiencies when only a small fraction of hyperparameters significantly contribute to the loss function. Random search, by sampling independently and uniformly across the entire space, is not subject to this curse of dimensionality and can more effectively locate regions that matter for model performance. Mathematically, random search has an additional advantage when the hyperparameters exhibit smooth or continuous relationships with the loss function. In this case, random search can probe the space probabilistically, discovering gradients of loss that grid search, due to its fixed grid structure, may miss. Furthermore, random search is capable of finding the optimum even when the loss function is non-convex, provided that the space is explored adequately. This becomes particularly relevant in the presence of highly irregular loss surfaces, as random search has the potential to escape local minima more effectively than grid search, which is constrained by its fixed sampling grid.

In conclusion, random search is a highly efficient and scalable approach for hyperparameter optimization in machine learning. By sampling hyperparameters from predefined probability distributions and evaluating the associated loss function, random search provides a computationally feasible method for high-dimensional hyperparameter spaces, outperforming grid search in many cases. Its probabilistic nature allows it to focus on relevant regions of the hyperparameter space, making it particularly advantageous when only a subset of hyperparameters significantly impacts the model's performance. As the number of iterations T increases, random search becomes more likely to converge to the optimal configuration, making it a powerful tool for hyperparameter tuning in complex models. The pros of Random search are:

1. More efficient than grid search, especially when some hyperparameters are less important.
2. Can explore a larger search space with fewer evaluations.

The cons of Random search are:

1. No guarantee of finding the optimal hyperparameters.
2. May still require many iterations for high-dimensional spaces.

15.4.3 Bayesian Optimization

Literature Review: Chang et. al. (2025) [414] applied Bayesian Optimization (BO) for hyperparameter tuning in machine learning models used for predicting landslide displacement. It explores the impact of BO in optimizing Support Vector Machines (SVM), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU), demonstrating how Bayesian techniques improve model accuracy and convergence rates. Cihan (2025) [415] used Bayesian Optimization to fine-tune XGBoost, LightGBM, Elastic Net, and Adaptive Boosting models for predicting biomass gasification output. The study finds that Bayesian Optimization outperforms Grid and Random Search in reducing computational overhead while improving predictive accuracy. Makomere et. al.

(2025) [416] integrated Bayesian Optimization for hyperparameter tuning in deep learning-based industrial process modeling. The study provides insights into how BO improves model generalization and reduces prediction errors in chemical process monitoring. Bakır (2025) [417] introduced TuneDroid, an automated Bayesian Optimization-based framework for hyperparameter tuning of Convolutional Neural Networks (CNNs) used in cybersecurity. The results suggest that Bayesian Optimization accelerates model training while improving malware detection accuracy. Khurshid et. al. (2025) [404] compared Bayesian Optimization and Random Search for tuning hyperparameters in XGBoost-based diabetes prediction models. It concludes that Bayesian Optimization provides a superior trade-off between speed and accuracy compared to traditional search methods. Liu et. al. (2025) [418] explored Bayesian Optimization's ability to fine-tune deep learning models for predicting acoustic performance in engineering systems. The authors demonstrate how Bayesian methods improve prediction accuracy while reducing computational costs. Balcan et. al. (2025) [411] provided a rigorous analysis of the sample complexity required for Bayesian Optimization in deep learning. The findings show that Bayesian Optimization requires fewer samples to converge to optimal solutions compared to other hyperparameter tuning techniques. Ma et. al. (2025) [419] integrated Bayesian Optimization with Support Vector Machines (SVMs) for anomaly detection in high-speed machining. They find that Bayesian Optimization allows more effective exploration of hyperparameter spaces, leading to improved model reliability. Bouzaidi et. al. (2025) [420] explored the impact of Bayesian Optimization on CNN-based models for image classification. It demonstrates how Bayesian techniques outperform traditional methods like Grid Search in transfer learning scenarios. Mustapha et. al. (2025) [421] integrated Bayesian Optimization for tuning a hybrid deep learning framework combining Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) for pneumonia detection. The results confirm that Bayesian Optimization enhances the efficiency of multi-model architectures in medical imaging.

Bayesian Optimization (BO) is a powerful, mathematically sophisticated method for optimizing complex, black-box objective functions, which is particularly useful in the context of hyperparameter tuning in machine learning models. These objective functions, denoted as $f : \mathcal{X} \to \mathbb{R}$, are often expensive to evaluate due to factors such as time-consuming training of models or noisy observations. In hyperparameter tuning, the objective function typically represents some performance metric of a machine learning model (e.g., accuracy, error, or loss) evaluated at specific hyperparameter configurations. The goal of Bayesian Optimization is to find the hyperparameter setting $\mathbf{x}^* \in \mathcal{X}$ that minimizes (or maximizes) the objective function, such that:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \tag{15.328}$$

Given that exhaustive search is computationally prohibitive, BO uses a probabilistic approach to efficiently explore the hyperparameter space. This is achieved by treating the objective function $f(\mathbf{x})$ as a random function and utilizing a **surrogate model** to approximate it, which allows for strategic decisions about which points in the space $\mathcal{X}$ to evaluate. The surrogate model is typically represented by a **Gaussian Process (GP)**, which provides both a prediction and an uncertainty estimate at any point in $\mathcal{X}$. The GP is a non-parametric, probabilistic model that assumes that function values at any finite set of points follow a joint Gaussian distribution. Specifically, for a set of observed points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the corresponding function values $\{f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)\}$ are assumed to be jointly distributed as:

$$\begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \tag{15.329}$$

where $\mathbf{m} = [m(\mathbf{x}_1), m(\mathbf{x}_2), \dots, m(\mathbf{x}_n)]^\top$ is the mean vector and $\mathbf{K}$ is the covariance matrix whose entries are defined by a covariance (or kernel) function $k(\mathbf{x}, \mathbf{x}')$, which encodes assumptions about the smoothness and periodicity of the objective function. The kernel function plays a crucial role in determining the properties of the Gaussian Process. A commonly used kernel is the **Squared Exponential (SE)** kernel, which is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right) \tag{15.330}$$

where σ_f^2 is the variance, which scales the function values, and ℓ is the length scale, which controls the smoothness of the function by dictating how quickly the function values can change with respect to the inputs. Once the Gaussian Process has been specified, Bayesian Optimization proceeds by updating the posterior distribution over the objective function after each new evaluation. Given a set of n observed pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ represents observational noise, we update the posterior of the GP to reflect the observed data. The posterior mean $\mu(\mathbf{x}_*)$ and variance $\sigma^2(\mathbf{x}_*)$ at a new point $\mathbf{x}_*$ are given by the following equations:

$$\mu(\mathbf{x}_*) = \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{y} \tag{15.331}$$

$$\sigma^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{k}_* \tag{15.332}$$

where $\mathbf{k}_*$ is the vector of covariances between the test point $\mathbf{x}_*$ and the observed points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, and $\mathbf{K}$ is the covariance matrix of the observed points. The updated mean $\mu(\mathbf{x}_*)$ provides the model's best guess for the value of the function at $\mathbf{x}_*$, and $\sigma^2(\mathbf{x}_*)$ quantifies the uncertainty associated with this estimate.

In Bayesian Optimization, the central objective is to select the next hyperparameter setting $\mathbf{x}_*$ to evaluate in such a way that the number of function evaluations is minimized while still making progress toward the global optimum. This is achieved by optimizing an **acquisition function**. The acquisition function $\alpha(\mathbf{x})$ represents a trade-off between exploiting regions of the input space where the objective function is expected to be low and exploring regions where the model's uncertainty is high. Several acquisition functions have been proposed, including **Expected Improvement (EI)**, **Probability of Improvement (PI)**, and **Upper Confidence Bound (UCB)**. The **Expected Improvement (EI)** acquisition function is one of the most widely used and is defined as:

$$\text{EI}(\mathbf{x}) = (f_{\text{best}} - \mu(\mathbf{x}))\Phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{15.333}$$

where f_{best} is the best observed value of the objective function, $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative distribution and probability density functions of the standard normal distribution, respectively, and $\sigma(\mathbf{x})$ is the standard deviation at $\mathbf{x}$. The first term measures the potential for improvement, weighted by the probability of achieving that improvement, and the second term reflects the uncertainty at $\mathbf{x}$, encouraging exploration in uncertain regions. The acquisition function is maximized at each iteration to select the next point $\mathbf{x}_*$:

$$\mathbf{x}_* = \arg\max_{\mathbf{x} \in \mathcal{X}} \text{EI}(\mathbf{x}) \tag{15.334}$$

An alternative acquisition function is the **Probability of Improvement (PI)**, which is simpler and directly measures the probability that the objective function at $\mathbf{x}$ will exceed the current best value:

$$\text{PI}(\mathbf{x}) = \Phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{15.335}$$

Another common acquisition function is the **Upper Confidence Bound (UCB)**, which balances exploration and exploitation by selecting the point with the highest upper confidence bound:

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}) \tag{15.336}$$

where κ is a hyperparameter that controls the trade-off between exploration (κ large) and exploitation (κ small). After selecting $\mathbf{x}_*$, the function is evaluated at this point, and the observed value $y_* = f(\mathbf{x}_*)$ is used to update the posterior distribution of the Gaussian Process. This process is repeated iteratively, and each new observation refines the model's understanding of the objective function, guiding the search for the optimal $\mathbf{x}^*$. One of the primary advantages of Bayesian Optimization is its ability to efficiently optimize expensive-to-evaluate functions by focusing the search on the most promising regions of the input space. However, as the number of observations increases, the computational complexity of maintaining the Gaussian Process model grows cubically with respect to the number of points, due to the need to invert the covariance matrix $\mathbf{K}$. This cubic complexity, $\mathcal{O}(n^3)$, can be prohibitive for large datasets. To mitigate this, techniques such as **sparse Gaussian Processes** have been developed, which approximate the full covariance matrix by using a smaller set of inducing points, thus reducing the computational cost while maintaining the flexibility of the Gaussian Process model.

In conclusion, Bayesian Optimization represents a mathematically rigorous and efficient method for hyperparameter tuning, where a Gaussian Process surrogate model is used to approximate the unknown objective function, and an acquisition function guides the search for the optimal solution by balancing exploration and exploitation. Despite its computational challenges, especially in high-dimensional problems, the method is widely applicable in contexts where evaluating the objective function is expensive, and it has been shown to outperform traditional optimization techniques in many real-world scenarios. The pros of Bayesian Optimization are:

1. Efficient and requires fewer evaluations compared to grid/random search.
2. Balances exploration (trying new regions) and exploitation (focusing on promising regions).

The cons of Bayesian Optimization are:

1. Computationally expensive to build and update the surrogate model.
2. May struggle with high-dimensional spaces or noisy objective functions.

15.4.4 Genetic Algorithms

Literature Review: Li et. al. [432] proposed a Genetic Algorithm-tuned deep transfer learning model for intrusion detection in IoT networks. The authors demonstrate that GA significantly enhances model generalization and efficiency by systematically optimizing network hyperparameters. Emmanuel et. al. (2025) [407] compared Genetic Algorithms, Bayesian Optimization, and Evolutionary Strategies for hyperparameter tuning of deep-learning models in protein interaction prediction. It highlights how GA efficiently explores large hyperparameter spaces, leading to faster convergence and better model performance. Gül and Bakır [433] developed GA-based optimization techniques for hyperparameter tuning in geophysical models. The authors demonstrate how GA significantly improves predictive accuracy in water conductivity modeling by effectively selecting optimal hyperparameters. Kalonia and Upadhyay (2025) [385] applied Genetic Algorithm-based tuning for CNN-RNN models in software fault prediction. The authors compare GA with Particle Swarm Optimization (PSO) and find that GA provides better robustness in feature selection and model optimization. Sen et. al. (2025) [434] explored a hybrid Genetic Algorithm-Particle Swarm Optimization (GA-PSO) approach to optimize QLSTM models for weather forecasting. The authors show that GA-based tuning enhances model adaptability in dynamic meteorological environments. Roy et. al. (2025) [435] integrated Genetic Algorithms with Bayesian Optimization to improve the diagnosis of glaucoma using deep learning. The study finds that GA helps in selecting hyperparameters that lead to more stable and interpretable medical AI models. Jiang et. al. (2025) [436] applied Genetic Algorithm hyperparameter tuning for machine learning models used in coastal drainage system optimization. The results indicate GA's ability to optimize models for real-world engineering applications where trial-and-error is costly. Borah and Chandrasekaran (2025) [437] applied Genetic Algorithm tuning to optimize machine learning models for predicting mechanical properties of 3D-printed materials. The authors highlight GA's ability to balance exploration and exploitation in hyperparameter tuning. Tan et. al. (2025) [438] integrated Genetic Algorithms with Reinforcement Learning for tuning hyperparameters in transportation models. The study finds that GA-based tuning reduces energy consumption while maintaining operational efficiency. Galindo et. al. (2025) [439] applied Multi-Objective Genetic Algorithms (MOGA) to hyperparameter tuning in fairness-aware machine learning models. The authors find that MOGA leads to balanced models that maintain predictive performance while minimizing bias.

Hyperparameter tuning in machine learning is fundamentally an optimization problem where the objective is to determine the best set of hyperparameters for a given model to achieve the lowest possible validation error or the highest possible performance metric. Mathematically, if we denote the hyperparameters as a vector

$$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_n), \tag{15.337}$$

where each λ_i belongs to a search space Λ_i, then the optimization problem can be formally written as

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}\in\Lambda} f(\boldsymbol{\lambda}) \tag{15.338}$$

where $f : \Lambda \to \mathbb{R}$ is an objective function, typically the validation loss of a machine learning model. This function is often **non-convex, non-differentiable, high-dimensional, and stochastic**, which makes conventional gradient-based methods inapplicable. Moreover, the search space Λ may consist of both continuous and discrete hyperparameters, further complicating the problem. Given the computational complexity of exhaustive search methods such as **grid search** and the inefficiency of purely random search methods, **Genetic Algorithms (GAs)** provide a heuristic but powerful optimization framework inspired by principles of natural evolution.

Genetic Algorithms belong to the class of **stochastic, population-based metaheuristic optimization methods**. They are designed to iteratively evolve a population of candidate solutions toward better solutions based on a fitness metric. Each iteration in a Genetic Algorithm is referred to as a **generation**, and the core operations that drive evolution include **selection, crossover, and mutation**. These operators collectively ensure that the algorithm explores and exploits the hyperparameter space efficiently, balancing between **global exploration** (to avoid local optima) and **local exploitation** (to refine promising solutions). Formally, at iteration t, the Genetic Algorithm maintains a **population** of hyperparameter candidates

$$\mathcal{P}_t = \{\boldsymbol{\lambda}_1^{(t)}, \boldsymbol{\lambda}_2^{(t)}, \dots, \boldsymbol{\lambda}_N^{(t)}\} \tag{15.339}$$

where N is the population size, and each individual $\boldsymbol{\lambda}_i^{(t)}$ is evaluated using an objective function f, yielding a fitness value

$$F_i^{(t)} = f(\boldsymbol{\lambda}_i^{(t)}). \tag{15.340}$$

The evolution of the population from generation t to $t + 1$ follows a structured process, beginning with **Selection**. The selection mechanism determines which hyperparameter candidates will serve as parents to generate offspring for the next generation. A commonly used selection method is **fitness-proportional selection**, also known as **roulette wheel selection**, where the probability of selecting an individual $\boldsymbol{\lambda}_i$ is given by

$$P(\boldsymbol{\lambda}_i) = \frac{e^{-\beta F_i}}{\sum_{j=1}^{N} e^{-\beta F_j}}. \tag{15.341}$$

Here, $\beta > 0$ controls the **selection pressure**, determining how much preference is given to high-performing individuals. If β is too high, selection is overly greedy and can lead to **premature**

convergence; if too low, selection becomes nearly random, reducing the convergence rate. This selection process ensures that better-performing hyperparameter configurations have a higher probability of propagating to the next generation while still allowing some stochastic diversity.

After selection, the next step is **Crossover**, also known as **recombination**, which involves combining the genetic information of two parents to produce offspring. Mathematically, given two parent hyperparameter vectors $\boldsymbol{\lambda}_A$ and $\boldsymbol{\lambda}_B$, a child $\boldsymbol{\lambda}_C$ is generated via a convex combination:

$$\lambda_{C,j} = \alpha \lambda_{A,j} + (1-\alpha)\lambda_{B,j}, \quad \alpha \sim \text{Uniform}(0,1). \tag{15.342}$$

This is known as **blend crossover**, which ensures a smooth interpolation between parent solutions. Other crossover techniques include **one-point crossover**, where a random split point k is chosen and the first k components come from one parent while the remaining components come from the other parent. The use of crossover ensures that useful information is inherited from multiple parents, promoting efficient exploration of the search space. To maintain diversity and prevent premature convergence, **Mutation** is applied, introducing small random perturbations to the offspring. Mathematically, this can be expressed as

$$\lambda_j^{\text{new}} = \lambda_j + \delta, \quad \delta \sim \mathcal{N}(0, \sigma^2), \tag{15.343}$$

where σ controls the mutation step size. In **adaptive genetic algorithms**, σ decreases over time:

$$\sigma_t = \sigma_0 e^{-\gamma t} \tag{15.344}$$

for some decay rate $\gamma > 0$, implementing **annealing-based exploration**, which helps refine solutions as the algorithm progresses. The convergence behavior of Genetic Algorithms can be analyzed through the **expected fitness improvement** formula:

$$\mathbb{E}[F^{(t+1)}] \leq \mathbb{E}[F^{(t)}] - \eta \cdot \text{Var}[F^{(t)}] \tag{15.345}$$

where η is a learning rate influenced by the mutation rate μ. This follows a **Lyapunov stability argument**, implying eventual convergence under bounded variance conditions. Additionally, Genetic Algorithms operate as a **Markov Chain**, satisfying:

$$P(\mathcal{P}_{t+1}|\mathcal{P}_t, \mathcal{P}_{t-1}, \dots) = P(\mathcal{P}_{t+1}|\mathcal{P}_t). \tag{15.346}$$

Thus, GAs approximate a **randomized hill-climbing process** with enforced diversity, ensuring a good tradeoff between **exploration and exploitation**. Genetic Algorithms offer significant advantages over traditional hyperparameter tuning methods. **Grid Search**, which evaluates all combinations exhaustively, suffers from exponential complexity $O(k^n)$ for n hyperparameters with k values each. **Random Search**, though more efficient, lacks any adaptation to previous evaluations. GAs, in contrast, leverage **historical information and evolutionary dynamics** to efficiently search the space while maintaining diversity.

In summary, Genetic Algorithms provide a **powerful, biologically inspired approach** to hyperparameter tuning, leveraging evolutionary principles to efficiently explore high-dimensional, non-convex, and discontinuous search spaces. Their combination of **selection, crossover, and mutation**, along with well-defined convergence properties, makes them highly effective in optimizing machine learning hyperparameters. The rigorous mathematical framework underlying GAs ensures that they are not merely heuristic methods but **robust, theoretically justified optimization algorithms** that can adapt dynamically to complex hyperparameter landscapes. The pros of Genetic Algorithms are:

1. Can explore a wide range of hyperparameter combinations.

2. Suitable for non-differentiable or discontinuous objective functions.

The cons of Genetic Algorithms are:

1. Computationally expensive and slow to converge.

2. Requires careful tuning of mutation and crossover parameters.

15.4.5 Hyperband

Literature Review: Li et. al. (2018) [486] introduced the HyperBand algorithm. It provides a theoretical foundation for HyperBand, demonstrating its efficiency in hyperparameter optimization by dynamically allocating resources to promising configurations. The authors rigorously analyze its performance compared to traditional methods like random search and Bayesian optimization, proving its superiority in terms of speed and scalability. Falkner et. al. (2018) [487] combined Bayesian Optimization (BO) with HyperBand (HB) to create BOHB, a hybrid method that leverages the strengths of both approaches. It introduces a robust and scalable framework for hyperparameter tuning, particularly effective for large-scale machine learning tasks. The authors provide extensive empirical evaluations, demonstrating BOHB's efficiency and robustness. Li et. al. (2020) [488] extended HyperBand to a distributed computing environment, enabling massively parallel hyperparameter tuning. The authors introduce a system architecture that scales HyperBand to thousands of workers, making it practical for large-scale industrial applications. The paper also provides insights into the trade-offs between resource allocation and optimization performance. While not exclusively about HyperBand, the paper by Snoek et. al. (2012) [489] laid the groundwork for understanding Bayesian optimization, which is often compared to HyperBand. It provides a comprehensive framework for hyperparameter tuning, which is useful for understanding the context in which HyperBand operates and its advantages over Bayesian methods. Slivkins et. al. (2024) [490] provided a thorough theoretical foundation for multi-armed bandit algorithms, which are the basis for HyperBand. It explains the principles of resource allocation and exploration-exploitation trade-offs, offering a deeper understanding of how HyperBand achieves efficient hyperparameter optimization. Hazan et. al. (2018) [491] explored spectral methods for hyperparameter optimization, providing a theoretical perspective that complements HyperBand's empirical approach. It discusses the limitations of traditional methods and highlights the advantages of bandit-based approaches like HyperBand. Domhan et. al. (2015) [492] introduced the concept of learning curve extrapolation, which is a key component of HyperBand's success. It demonstrates how early stopping and resource allocation can be optimized by predicting the performance of hyperparameter configurations, a technique that HyperBand later formalizes and extends. Agrawal (2021) [493] provided a comprehensive overview of hyperparameter optimization techniques, including a detailed chapter on HyperBand. It explains the algorithm's mechanics, its advantages over other methods, and practical implementation tips. The book is particularly useful for practitioners looking to apply HyperBand in real-world scenarios. Shekhar et. al. (2021) [494] compared various hyperparameter optimization tools, including HyperBand, Bayesian optimization, and random search. It provides empirical evidence of HyperBand's efficiency and scalability, particularly for large datasets and complex models. The paper also discusses the trade-offs between different methods. Bergstra et. al. (2011) [495] discussed the challenges of hyperparameter optimization in neural networks and introduces early methods for addressing them. While it predates HyperBand, it provides valuable context for understanding the evolution of hyperparameter optimization techniques and the need for more efficient methods like HyperBand.

Let Λ denote the hyperparameter space, and let $\lambda \in \Lambda$ be a hyperparameter configuration. The goal is to minimize a loss function $L(\lambda)$, which is evaluated using a validation set or cross-validation. The evaluation of $L(\lambda)$ is computationally expensive, as it typically involves training a model and computing its performance. We assume:

- $L(\lambda)$ is a black-box function with no known analytical form.
- Evaluating $L(\lambda)$ with a budget b (e.g., number of epochs, dataset size) yields an approximation $L(\lambda, b)$, where $L(\lambda, b) \to L(\lambda)$ as $b \to R$, and R is the maximum budget.

HyperBand relies on the following assumptions for its theoretical guarantees: For any λ, $L(\lambda, b)$ is non-increasing in b. That is, increasing the budget improves performance:

$$b_1 \leq b_2 \implies L(\lambda, b_1) \geq L(\lambda, b_2). \tag{15.347}$$

The maximum budget R is finite, and $L(\lambda, R) = L(\lambda)$. There exists a unique optimal configuration $\lambda^* \in \Lambda$ such that:

$$L(\lambda^*) \leq L(\lambda), \quad \forall \lambda \in \Lambda. \tag{15.348}$$

Successive Halving is the Building Block of HyperBand Method. HyperBand generalizes the Successive Halving (SH) algorithm. SH operates as follows:

1. Start with n configurations and allocate a small budget b to each.
2. Evaluate all configurations and keep the top $1/\eta$ fraction.
3. Increase the budget by a factor of η and repeat until one configuration remains.

The total cost of SH is:

$$C_{SH} = \sum_{i=0}^{s-1} n_i \cdot b_i, \tag{15.349}$$

where $n_i = \frac{n}{\eta^i}$ and $b_i = b \cdot \eta^i$. HyperBand introduces a bracket-based approach to explore different trade-offs between n (number of configurations) and b (budget per configuration). It consists of two nested loops: Outer Loop and Inner Loop. For Outer Loop, For each bracket $s \in \{0, 1, \ldots, s_{\max}\}$ we have to compute the number of configurations n and the initial budget b:

$$n = \left\lfloor \frac{s_{\max} + 1}{s + 1} \cdot \eta^s \right\rfloor, \quad b = R \cdot \eta^{-s}. \tag{15.350}$$

Here, $s_{\max} = \lfloor \log_\eta(R) \rfloor$ is the number of brackets. We have to run the Inner Loop (Successive Halving) with n configurations and initial budget b. For Inner Loop (Successive Halving), we have to first randomly sample n configurations $\lambda_1, \ldots, \lambda_n$. For each round $i \in \{0, 1, \ldots, s\}$:

- Allocate budget $b_i = b \cdot \eta^i$ to each configuration.
- Evaluate $L(\lambda_j, b_i)$ for all j.
- Keep the top $n_i = \frac{n}{\eta^i}$ configurations based on $L(\lambda_j, b_i)$.

Return the best configuration from the final round. HyperBand's efficiency stems from its ability to explore multiple resource allocation strategies. Below, we analyze its properties rigorously. The total cost of HyperBand is the sum of costs across all brackets:

$$C_{HB} = \sum_{s=0}^{s_{\max}} C_{SH}(s), \tag{15.351}$$

where $C_{SH}(s)$ is the cost of Successive Halving in bracket s. HyperBand balances exploration and exploitation by varying s:

- For small s, it explores many configurations with small budgets.
- For large s, it exploits fewer configurations with large budgets.

This ensures that HyperBand does not prematurely discard potentially optimal configurations. Under the assumptions of monotonicity and finite budget, HyperBand achieves the following:

- **Near-Optimality:** The best configuration found by HyperBand converges to λ^* as $R \to \infty$.
- **Logarithmic Scaling:** The total cost C_{HB} scales logarithmically with the number of configurations.

We sketch a proof of HyperBand's efficiency under the given assumptions. By monotonicity, the ranking of configurations improves as the budget increases. Thus, the top configurations in early rounds are likely to include λ^*. The cost of each bracket s is:

$$C_{SH}(s) = \sum_{i=0}^{s} n_i \cdot b_i = \sum_{i=0}^{s} \frac{n}{\eta^i} \cdot b \cdot \eta^i = n \cdot b \cdot (s+1). \tag{15.352}$$

Substituting n and b from the outer loop:

$$C_{SH}(s) = \left\lfloor \frac{s_{\max}+1}{s+1} \cdot \eta^s \right\rfloor \cdot R \cdot \eta^{-s} \cdot (s+1). \tag{15.353}$$

For large $s_{\max}$, this simplifies to:

$$C_{SH}(s) \approx R \cdot (s_{\max}+1). \tag{15.354}$$

Thus, the total cost C_{HB} scales as:

$$C_{HB} \approx R \cdot (s_{\max}+1)^2. \tag{15.355}$$

Since $s_{\max} = \lfloor \log_\eta(R) \rfloor$, the cost scales logarithmically with R. There are some impressive practical implications of HyperBand Method. HyperBand's theoretical guarantees make it highly effective for:

- **Large-Scale Optimization:** It scales to high-dimensional hyperparameter spaces.
- **Parallelization:** Configurations can be evaluated independently, enabling distributed computation.
- **Adaptability:** It works for both continuous and discrete hyperparameter spaces.

In conclusion, HyperBand is a mathematically rigorous and efficient algorithm for hyperparameter optimization. By generalizing Successive Halving and exploring multiple resource allocation strategies, it achieves a near-optimal balance between exploration and exploitation.

15.4.6 Gradient-Based Optimization

Literature Review: Snoek et. al. (2012) [489] introduced Bayesian optimization as a powerful framework for hyperparameter tuning. While not strictly gradient-based, it lays the foundation for gradient-based methods by emphasizing the importance of efficient search strategies in high-dimensional spaces. It also discusses the use of Gaussian processes for modeling the hyperparameter response surface, which can be combined with gradient-based techniques. Maclaurin et. al. (2015) [497] introduced a novel method for gradient-based hyperparameter optimization by making the learning process reversible. It allows gradients of the validation loss with respect to hyperparameters to be computed efficiently, enabling the use of gradient descent for hyperparameter tuning. This approach is particularly effective for tuning continuous hyperparameters. Pedregosa et. al. (2016) [498] proposed a gradient-based method for hyperparameter optimization that uses an approximate gradient computed through implicit differentiation. It is particularly useful for large-scale problems and provides a theoretical framework for understanding the convergence properties of gradient-based hyperparameter optimization. Franceschi et. al. (2017) [500] compared forward-mode and reverse-mode automatic differentiation for hyperparameter optimization. It provides insights into the computational trade-offs between these methods and demonstrates their effectiveness in tuning hyperparameters for deep learning models. While primarily focused on neural architecture search (NAS), this paper by Zoph (2016) [496] introduced gradient-based methods for optimizing hyperparameters in the context of reinforcement learning. It demonstrates how gradient-based optimization can be applied to discrete and continuous hyperparameters in complex search spaces. Hazan et. al. (2018) [491] proposed a spectral approach to hyperparameter optimization, leveraging gradient-based methods to optimize hyperparameters in a low-dimensional subspace. It provides theoretical guarantees for convergence and demonstrates practical improvements in tuning efficiency. Bergstra et. al. (2011) [495] explored the use of gradient-based methods for hyperparameter optimization in neural networks. It highlights the challenges of applying gradient-based methods to discrete hyperparameters and proposes solutions for handling such cases. Franceschi et. al. (2018) [499] formalized hyperparameter optimization as a bilevel programming problem and proposes gradient-based methods to solve it. It provides a unified framework for understanding hyperparameter optimization and meta-learning, with applications to both continuous and discrete hyperparameters. Liu et. al. (2019) [501] introduced a differentiable architecture search (DARTS) method that uses gradient-based optimization to tune hyperparameters in neural architectures. It significantly reduces the computational cost of architecture search and demonstrates the effectiveness of gradient-based methods in complex search spaces. Lorraine et. al. (2020) [502] introduced a scalable method for gradient-based hyperparameter optimization using implicit differentiation. It enables the optimization of millions of hyperparameters efficiently, making it suitable for large-scale machine learning models. The paper also provides theoretical insights into the convergence properties of the method.

In practical learning problems, we optimize over a **function space** rather than a finite-dimensional vector space. Define:

- **Hypothesis space**: $\mathcal{H}$ as a **Banach space** equipped with norm $\|\cdot\|_{\mathcal{H}}$.
- **Parameter space**: $\Theta \subseteq \mathcal{H}$, where Θ is a closed, convex subset of $\mathcal{H}$.

We optimize:

$$\theta^*(\lambda) = \arg\min_{\theta \in \Theta} \mathcal{L}_{train}(\theta, \lambda) \tag{15.356}$$

where $\mathcal{L}_{train}$ is a Fréchet differentiable function on $\mathcal{H}$. By Inner Product Structure in Hilbert Spaces, if $\mathcal{H}$ is a **Hilbert space**, then there exists an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, which induces a norm:

$$\|\theta\|_{\mathcal{H}} = \sqrt{\langle \theta, \theta \rangle_{\mathcal{H}}} \tag{15.357}$$

The optimization problem is now posed in a **functional setting**. Using Variational Formulation of Hyperparameter Optimization, Instead of solving a constrained minimization, we express the optimization problem using the **Euler-Lagrange equation**. The hyperparameter tuning problem is:

$$\lambda^* = \arg\min_{\lambda} \mathcal{F}(\lambda) \tag{15.358}$$

where:

$$\mathcal{F}(\lambda) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{val}} \left[\mathcal{L}_{val}(\theta^*(\lambda), \lambda)\right] \tag{15.359}$$

Since $\theta^*(\lambda)$ is the minimizer of $\mathcal{L}_{train}$, it satisfies the **Euler-Lagrange equation**:

$$\frac{\delta \mathcal{L}_{train}}{\delta \theta}(\theta^*(\lambda), \lambda) = 0 \tag{15.360}$$

To differentiate $\mathcal{F}(\lambda)$, apply the **chain rule in variational calculus**:

$$\frac{d}{d\lambda}\mathcal{F}(\lambda) = \frac{\partial \mathcal{L}_{val}}{\partial \lambda} + \left\langle \frac{\delta \mathcal{L}_{val}}{\delta \theta}, \frac{d\theta^*}{d\lambda} \right\rangle_{\mathcal{H}} \tag{15.361}$$

Applying the **second-order Gateaux derivative**:

$$\frac{d\theta^*}{d\lambda} = -\left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda \delta \theta} \tag{15.362}$$

Substituting, we get the **hyperparameter gradient**:

$$\nabla_\lambda \mathcal{F}(\lambda) = \frac{\partial \mathcal{L}_{val}}{\partial \lambda} - \left\langle \frac{\delta \mathcal{L}_{val}}{\delta \theta}, \left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda \delta \theta} \right\rangle_{\mathcal{H}} \tag{15.363}$$

We should now do the Higher-Order Sensitivity Analysis. Beyond first and second derivatives, we analyze third-order terms using **Taylor expansions in Banach spaces**:

$$\theta^*(\lambda + \Delta\lambda) = \theta^*(\lambda) + \frac{d\theta^*}{d\lambda}\Delta\lambda + \frac{1}{2}\frac{d^2\theta^*}{d\lambda^2}(\Delta\lambda)^2 + O(\|\Delta\lambda\|^3) \tag{15.364}$$

The second-order sensitivity term is:

$$\frac{d^2\theta^*}{d\lambda^2} = -\left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \left[\frac{\delta^3 \mathcal{L}_{train}}{\delta \lambda \delta \theta^2}\frac{d\theta^*}{d\lambda} + \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda^2 \delta \theta}\right] \tag{15.365}$$

Thus, the second-order expansion of the hyperparameter function is:

$$\mathcal{F}(\lambda + \Delta\lambda) = \mathcal{F}(\lambda) + \nabla_\lambda \mathcal{F}(\lambda)\Delta\lambda + \frac{1}{2}\Delta\lambda^\top \nabla^2_{\lambda\lambda}\mathcal{F}(\lambda)\Delta\lambda + O(\|\Delta\lambda\|^3) \tag{15.366}$$

By Spectral Analysis of Hessians, The Hessian $H = \nabla^2_{\theta\theta}\mathcal{L}_{train}$ governs curvature. We perform **eigenvalue decomposition**:

$$H = Q\Lambda Q^\top, \quad \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p) \tag{15.367}$$

If $\lambda_{\min} > 0$, H is **positive definite**, ensuring local convexity and If $\lambda_{\min} = 0$, H is **singular**, requiring pseudo-inversion. Using **Tikhonov regularization**, we modify:

$$H_\epsilon = H + \epsilon I, \quad \text{where } \epsilon > 0 \tag{15.368}$$

Then, the modified inverse is:

$$H_\epsilon^{-1} = Q\Lambda_\epsilon^{-1}Q^\top, \quad \Lambda_\epsilon^{-1} = \operatorname{diag}\left(\frac{1}{\lambda_1 + \epsilon}, \ldots, \frac{1}{\lambda_p + \epsilon}\right) \tag{15.369}$$

This prevents numerical instability. From a Manifold Perspective we have to do Optimization on Riemannian Spaces. Instead of optimizing in $\mathbb{R}^p$, let Θ be a **Riemannian manifold** with metric g. The update rule becomes:

$$\lambda_{t+1} = \operatorname{Exp}_{\lambda_t}\left(-\eta g_{\lambda_t}^{-1}\nabla_\lambda \mathcal{F}(\lambda_t)\right) \tag{15.370}$$

where $\operatorname{Exp}_\lambda(\cdot)$ is the **Riemannian exponential map**. In conclusion, this analysis extends hyperparameter tuning to **functional spaces**, introducing **variational methods, higher-order derivatives, spectral analysis, and Riemannian optimization**.

15.4.7 Population-Based Training (PBT)

Literature Review: Jaderberg et al. (2017) [504] introduced Population-Based Training (PBT). It combines the strengths of random search and hand-tuning by maintaining a population of models that are trained in parallel. The key innovation is the use of exploitation (copying weights from better-performing models) and exploration (perturbing hyperparameters) to dynamically optimize hyperparameters during training. The paper demonstrates PBT's effectiveness on deep reinforcement learning and supervised learning tasks. Liang et. al. (2017) [503] provided a comprehensive analysis of population-based methods for hyperparameter optimization in deep learning. It compares PBT with other evolutionary algorithms and highlights its advantages in terms of computational efficiency and adaptability. The authors also discuss practical considerations for implementing PBT in large-scale training scenarios. Co-Reyes et. al. [505] explored the use of PBT for meta-optimization, specifically for evolving reinforcement learning algorithms. It demonstrates how PBT can be used to discover novel RL algorithms by optimizing both hyperparameters and algorithmic components. The work shows PBT's versatility beyond standard hyperparameter tuning. Song et. al. (2024) [506] applied PBT to Neural Architecture Search (NAS), showing how PBT can efficiently explore and exploit architectures and hyperparameters simultaneously. It provides insights into how PBT can reduce the computational cost of NAS while maintaining competitive performance. Wan et. al. (2022) [507] bridged the gap between Bayesian Optimization (BO) and PBT by proposing a hybrid approach. It uses BO to guide the initial hyperparameter search and PBT to refine hyperparameters dynamically during training. The paper demonstrates improved performance over standalone PBT or BO. Garcia-Valdez et. al. (2023) [508] addressed the scalability of PBT in distributed computing environments. It introduces an asynchronous variant of PBT that reduces idle time and improves resource utilization. The work is particularly relevant for large-scale machine-learning applications.

Let's do the Mathematical Formulation of PBT: Dynamic Hyperparameter Optimization. For that let us denote the population of models at time t as $\mathcal{P}(t) = \{(\theta_i, h_i)\}_{i=1}^N$, where:

- $\theta_i \in \mathbb{R}^d$ represents the model parameters, with d being the dimensionality of the model parameter space.
- $h_i \in \mathbb{H} \subset \mathbb{R}^m$ represents the hyperparameters of the i-th model, with m being the dimensionality of the hyperparameter space $\mathcal{H}$. The set $\mathcal{H}$ is a bounded subset of the positive real numbers, such as learning rates, batch sizes, or regularization factors.

We have to now use the Loss Function as a Metric. The objective function $\mathcal{L}(\theta, h)$ is a mapping from the space of model parameters and hyperparameters to a scalar loss value. This loss function is a non-convex, potentially non-differentiable function in high-dimensional spaces, particularly in the context of deep neural networks.

$$\mathcal{L}(\theta, h) = \mathcal{L}_{\text{train}}(\theta, h) + \mathcal{L}_{\text{val}}(\theta, h) \tag{15.371}$$

where $\mathcal{L}_{\text{train}}(\theta, h)$ is the training loss, and $\mathcal{L}_{\text{val}}(\theta, h)$ is the validation loss. Here, $\mathcal{L}_{\text{val}}$ serves as the fitness function upon which the hyperparameter optimization process is based. Using the Exploitation-Exploration Framework, the central mechanism of PBT revolves around two processes: **exploitation** (model selection) and **exploration** (hyperparameter mutation). We will delve into these components through the lens of **Markov Decision Processes (MDPs)**, optimization theory, and stochastic calculus. Regarding the Selection Mechanism (Exploitation), the models in the population are ranked based on their validation fitness $M_i(t)$ at each time step t:

$$M_i(t) = \mathcal{L}_{\text{val}}(\theta_i, h_i) \tag{15.372}$$

This ranking corresponds to a sorted order:

$$M_1(t) \geq M_2(t) \geq \cdots \geq M_N(t) \tag{15.373}$$

In terms of **selection**, the worst-performing models are replaced by the best-performing models. We now formally express the selection step in terms of the updating mechanism. Given a population of models $\mathcal{P}(t)$, at time step t, a new model $\theta_i(t+1), h_i(t+1)$ inherits its hyperparameters $h_i(t)$ and model parameters $\theta_i(t)$ from the best-performing models, denoted by i^*. Thus, the hyperparameter update rule for the next iteration is:

$$h_i(t+1) = h_{i^*}(t), \quad \theta_i(t+1) = \theta_{i^*}(t) \tag{15.374}$$

This corresponds to the **exploitation** phase, where we take the best-performing hyperparameters from the current generation to seed the next. Regarding the Mutation Mechanism (Exploration), the mutation process injects randomness into the hyperparameters to encourage exploration of the search space. To formally describe this process, we use a stochastic perturbation model. Let $h_i(t)$ be the hyperparameters at time t. Mutation introduces a random perturbation to the hyperparameters as:

$$h_i(t+1) = h_i(t) \cdot (1 + \epsilon_i(t)) \tag{15.375}$$

where $\epsilon_i(t) \sim \mathcal{U}(-\alpha, \alpha)$ represents a random perturbation drawn from a uniform distribution with parameter α. This random perturbation ensures that the hyperparameters can adaptively escape local minima, promoting a more global search in the hyperparameter space. The mutative process can be seen as:

$$h_i(t+1) = h_i(t) \cdot 10^{\mathcal{U}(-\alpha,\alpha)} \tag{15.376}$$

This mutation process is a continuous stochastic process with a bounded magnitude, facilitating a fine balance between **exploitation** and **exploration**. We now interpret PBT as a **non-stationary**, **stochastic optimization** problem with dynamic model parameter and hyperparameter updates. In optimization terms, PBT involves iteratively optimizing a non-convex function $\mathcal{L}(\theta, h)$ with respect to the hyperparameters h, and the model parameters θ. The stochastic update for $h_i(t)$ can be modeled as:

$$h_i(t+1) = h_i(t) + \nabla_h \mathcal{L}(\theta_i(t), h_i(t)) + \sigma \cdot \mathcal{N}(0, I) \tag{15.377}$$

where $\nabla_h \mathcal{L}(\theta_i(t), h_i(t))$ is the gradient of the loss function with respect to the hyperparameters h_i, representing the exploitation mechanism (steepest descent direction), $\mathcal{N}(0, I)$ is a noise term with zero mean and identity covariance matrix, modeling the exploration mechanism, σ is a hyperparameter that controls the magnitude of the noise, thus influencing the exploration rate. We shall now do th Convergence Analysis via Lyapunov Stability. To rigorously analyze the convergence of PBT, we leverage **Lyapunov's stability theory**, which provides insight into whether the system of updates stabilizes or diverges. Define the **Lyapunov function** $V(t)$, which represents the deviation from the optimal solution h^* in terms of squared Euclidean distance:

$$V(t) = \sum_{i=1}^{N} \|h_i(t) - h^*\|^2 \tag{15.378}$$

The evolution of $V(t)$ over time gives us information about the behavior of the hyperparameters as the population evolves. If the system converges to a local optimum, we expect that $\mathbb{E}[V(t+1)] < V(t)$. Using the update rule for $h_i(t)$, we can compute the expected rate of change of the Lyapunov function:

$$\mathbb{E}[V(t+1) - V(t)] = -\delta V(t) \tag{15.379}$$

where $\delta > 0$ is a constant that guarantees exponential convergence towards the optimal hyperparameter configuration. This exponential decay implies that the population of models is moving toward a global optimum at a rate proportional to the current deviation from the optimal solution. Regarding the Generalized Stochastic Optimization Framework, PBT can be viewed as an instance of **stochastic optimization** under **non-stationary conditions**. The optimization process evolves by sequentially adjusting the hyperparameters and parameters according to a noisy gradient update:

$$h_i(t+1) = h_i(t) + \eta(t) \cdot (\nabla_h \mathcal{L}(\theta_i(t), h_i(t)) + \epsilon_i(t)) \tag{15.380}$$

Here $\eta(t)$ is a learning rate that decays over time, ensuring that the updates become smaller as the optimization progresses. The term $\epsilon_i(t)$ introduces noise for **exploration**, and the gradient term $\nabla_h \mathcal{L}$ ensures that the system **exploits** the current state of the model for refinement. Regarding the Theoretical Convergence Guarantees, Under appropriate conditions, PBT guarantees that the models will converge to an optimal or near-optimal hyperparameter configuration. By applying **perturbation theory** and **large deviation principles**, we can demonstrate that the population converges to a near-optimal region of the hyperparameter space with high probability. Furthermore, as $N \to \infty$, the convergence rate improves, which underscores the efficiency of the **population-based** approach in exploring high-dimensional hyperparameter spaces. Regarding Computational Efficiency and Parallelism in PBT, One of the key advantages of PBT is its **parallelizability**. Since each model in the population is trained independently, the process is well-suited to modern distributed computing environments, such as **multi-GPU** or **multi-TPU** setups. The time complexity of the population-based optimization process can be analyzed as follows:

- At each iteration t, we perform:
  - N forward passes to compute the losses $\mathcal{L}_{\text{val}}(\theta_i(t), h_i(t))$.
  - N selection and mutation operations for updating the population.
- This leads to a time complexity of $O(N)$ per iteration.

Since each model is evaluated independently, this process can be easily parallelized, allowing for significant speedup in hyperparameter optimization, particularly when the number of models in the population is large.

15.4.8 Optuna

Literature Review: Akiba et. al. (2019) [509] wrote the foundational paper introducing Optuna. It describes the framework's design principles, including its define-by-run API, efficient sampling algorithms, and pruning mechanisms. The paper highlights Optuna's scalability and flexibility compared to other hyperparameter optimization tools like Hyperopt and Bayesian Optimization. Kadhim et. al. (2022) [511] provided a comprehensive overview of hyperparameter optimization techniques, including Bayesian optimization, evolutionary algorithms, and bandit-based methods. It contextualizes Optuna within the broader landscape of hyperparameter tuning tools and methodologies. Bergstra et. al. (2011) [495] introduced the concept of sequential model-based optimization (SMBO) and tree-structured Parzen estimators (TPE), which are foundational to Optuna's sampling algorithms. It provides theoretical insights into efficient hyperparameter search strategies. Snoek et. al. (2012) [489] introduced Bayesian optimization using Gaussian processes (GPs) for hyperparameter tuning. While Optuna primarily uses TPE, this work is critical for understanding the theoretical underpinnings of probabilistic modeling in hyperparameter optimization. Akiba et. al. (2025) [510] expanded on the original Optuna paper, providing deeper insights into its define-by-run paradigm, which allows users to dynamically construct search spaces. It also discusses advanced features like multi-objective optimization and distributed computing. Yang and Shami (2020) [513] wrote a book that includes a practical guide to hyperparameter tuning, with examples using Optuna. It emphasizes the importance of tuning in deep learning and provides hands-on code snippets for integrating Optuna with Keras and TensorFlow. Wang (2024) [514] explained Optuna's support for multi-objective optimization, which is crucial for tasks like balancing model accuracy and computational cost. It provides practical examples and benchmarks. Frazier (2018) [515] provided a thorough introduction to Bayesian optimization, which is closely related to Optuna's TPE algorithm. It covers acquisition functions, Gaussian processes, and practical considerations for implementation. Jeba (2021) [512] wrote a collection of case studies that demonstrated Optuna's application in real-world scenarios, including hyperparameter tuning for deep learning, reinforcement learning, and time-series forecasting. It highlights Optuna's efficiency and ease of use. Hutter et. al. (2019) [516] provided a comprehensive overview of automated machine learning (AutoML), including hyperparameter optimization. It discusses Optuna in the context of AutoML frameworks and compares it with other tools like Auto-sklearn and TPOT.

Hyperparameter tuning, in the context of machine learning, is fundamentally an optimization problem defined over a hyperparameter space $\mathcal{H}$, which is typically a high-dimensional and heterogeneous domain comprising continuous, discrete, and categorical variables. Formally, let

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_n \tag{15.381}$$

where each $\mathcal{H}_i$ represents the domain of the i-th hyperparameter. The objective is to identify the optimal hyperparameter configuration $\mathbf{h}^* \in \mathcal{H}$ that minimizes (or maximizes) a predefined objective function

$$f : \mathcal{H} \to \mathbb{R} \tag{15.382}$$

which quantifies the performance of a machine learning model, such as validation loss or accuracy. Mathematically, this is expressed as

$$\mathbf{h}^* = \arg\min_{\mathbf{h} \in \mathcal{H}} f(\mathbf{h}) \tag{15.383}$$

The function $f(\mathbf{h})$ is often expensive to evaluate, as it requires training and validating a model, and is typically non-convex, noisy, and lacks an analytical gradient, rendering traditional optimization methods ineffective.

Optuna addresses this challenge by employing a Bayesian optimization framework, which iteratively constructs a probabilistic surrogate model of the objective function $f(\mathbf{h})$ and uses it to guide the search for $\mathbf{h}^*$. Specifically, Optuna utilizes a Tree-structured Parzen Estimator (TPE) as its surrogate model, which is a non-parametric density estimator that models the distribution of hyperparameters conditioned on the observed values of $f(\mathbf{h})$. The TPE approach partitions the observed trials into two subsets: G_{good}, containing hyperparameter configurations associated with the best observed values of $f(\mathbf{h})$, and G_{bad}, containing the remaining configurations. It then estimates two probability density functions,

$$p(\mathbf{h} \mid G_{\text{good}}) \quad \text{and} \quad p(\mathbf{h} \mid G_{\text{bad}}) \tag{15.384}$$

which represent the likelihood of hyperparameters given good and bad performance, respectively. The acquisition function $a(\mathbf{h})$, defined as the ratio

$$a(\mathbf{h}) = \frac{p(\mathbf{h} \mid G_{\text{good}})}{p(\mathbf{h} \mid G_{\text{bad}})} \tag{15.385}$$

is maximized to select the next hyperparameter configuration $\mathbf{h}_{\text{next}}$, thereby balancing exploration and exploitation in the search process. The optimization process begins with an initial phase of random sampling to build a preliminary model of $f(\mathbf{h})$, after which the TPE algorithm refines its probabilistic model and focuses on regions of $\mathcal{H}$ that are more likely to contain $\mathbf{h}^*$. This adaptive sampling strategy ensures that the search is both efficient and effective, particularly in high-dimensional spaces where the curse of dimensionality would otherwise render exhaustive search methods intractable. Additionally, Optuna incorporates pruning mechanisms to further enhance computational efficiency. Pruning involves terminating trials that are unlikely to yield improvements in $f(\mathbf{h})$ based on intermediate evaluations, thereby reducing the computational cost associated with unpromising configurations. This is achieved by comparing the performance of a trial at a given step to the performance of other trials at the same step and applying a statistical criterion to decide whether to continue or halt the trial. The convergence properties of Optuna's optimization process are grounded in the theoretical foundations of Bayesian optimization and TPE. Under mild assumptions, such as the smoothness of $f(\mathbf{h})$ and the proper calibration of the acquisition function, the algorithm is guaranteed to converge to the global optimum $\mathbf{h}^*$ as the number of trials N approaches infinity. However, in practice, the rate of convergence depends on the dimensionality of $\mathcal{H}$, the noise level of $f(\mathbf{h})$, and the efficiency of the surrogate model in capturing the underlying structure of the objective function. Optuna's implementation also supports advanced features such as conditional hyperparameter spaces, where the domain of one hyperparameter may depend on the value of another, and parallelization, which enables distributed evaluation of trials across multiple computational nodes.

In summary, Optuna provides a rigorous and mathematically sound framework for hyperparameter tuning by leveraging Bayesian optimization, TPE, and pruning mechanisms. Its ability to efficiently navigate complex and high-dimensional hyperparameter spaces, combined with its theoretical guarantees of convergence, makes it a powerful tool for optimizing machine learning models. The framework's flexibility, scalability, and integration with modern machine learning pipelines further enhance its utility in both research and practical applications. By formalizing hyperparameter tuning as a probabilistic optimization problem and employing advanced sampling and pruning strategies, Optuna achieves a balance between computational efficiency and optimization performance, ensuring that the identified hyperparameter configuration $\mathbf{h}^*$ is both optimal and robust.

15.4.9 Successive Halving

Literature Review: Egele et. al. (2024) [536] investigated an aggressive early stopping strategy for hyperparameter tuning in neural networks using Successive Halving. It compares standard SHA with learning curve extrapolation (LCE) and LC-PFN models, showing that early discarding significantly reduces computational costs while preserving model performance. Wojciuk et. al. (2024) [537] systematically compared different hyperparameter optimization methods, including Asynchronous Successive Halving (ASHA), Bayesian Optimization, and Grid Search, in tuning CNN models. It highlights the efficiency of ASHA in reducing the search space without sacrificing classification accuracy. Geissler et. al. (2024) [538] proposed an energy-efficient version of SHA called SM2. Their method adapts the Successive Halving process to reduce redundant energy-intensive training cycles, particularly beneficial for resource-constrained computing environments. Sarcheshmeh et. al. (2024) [539] applied SHA in engineering contexts, demonstrating how it optimizes hyperparameters in machine learning models for predicting concrete compressive strength. It provides insights into SHA's performance in structured regression problems. Sankar et. al. (2024) [540] applied Asynchronous Successive Halving (ASHA) for medical image analysis. It combines ASHA with PNAS (Progressive Neural Architecture Search) to improve disease classification, demonstrating SHA's capability in complex feature selection tasks. Zhang and Duh (2024) [541] rigorously examined how SHA can be optimized for neural machine translation models. It provides detailed experimental insights into how different configurations of SHA influence translation accuracy and computational efficiency. Aach et. al. (2024) [542] extended SHA by incorporating a "successive doubling" approach, dynamically adjusting resource allocation based on dataset size. This method improves performance when tuning models on high-performance computing (HPC) clusters. Jang et. al. (2024) [543] introduced QHB+, an optimization framework integrating SHA for automatic tuning of Spark SQL queries. It demonstrates how SHA can efficiently allocate computational resources in data-intensive applications. Chen et. al. (2024) [544] refined SHA's exploration-exploitation balance by integrating it with multi-armed bandit techniques. It evaluates different strategies for pruning underperforming hyperparameter configurations to accelerate optimization. Zhang et. al. (2024) [545] proposed FlexHB that extended SHA by introducing GloSH, an improved version of Successive Halving that dynamically adjusts resource allocation. The study highlights its advantages in reducing wasted computational resources while maintaining high-quality hyperparameter selection.

Hyperparameter optimization is a fundamental problem in machine learning, requiring the identification of an optimal configuration λ^* within a given search space Λ that minimizes a prescribed objective function. Mathematically, this optimization problem is formulated as the minimization of an expectation over the joint probability distribution of training and validation datasets, i.e.,

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathbb{E}_{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}} \left[\mathcal{L}(M(\lambda), \mathcal{D}_{\text{val}}) \right] \tag{15.386}$$

where $M(\lambda)$ is the machine learning model trained using hyperparameters λ, and $\mathcal{L}(\cdot)$ represents a loss function such as cross-entropy loss, mean squared error, or negative log-likelihood. Due to the large cardinality of Λ and the computational expense of evaluating $\mathcal{L}(M(\lambda), \mathcal{D}_{\text{val}})$, exhaustive evaluation of all configurations is infeasible. To mitigate this computational burden, **Successive Halving (SH)** is employed as a multi-fidelity optimization technique that dynamically allocates computational resources to promising candidates while progressively eliminating inferior configurations in a statistically justified manner.

The Successive Halving algorithm proceeds in a sequence of K iterative stages, where each stage consists of training, evaluation, ranking, and pruning of hyperparameter configurations. Let N denote the initial number of hyperparameter candidates sampled from Λ, and let B denote the total computational budget. The algorithm initializes each configuration with a budget of B_0 such that the sum of allocated budgets across all iterations remains bounded by B. Specifically, defining the **reduction factor** $\eta > 1$, the number of surviving configurations at each iteration is recursively defined as $N_k = N/\eta^k$, while the budget allocated to each surviving configuration follows the exponential growth pattern $B_k = \eta B_{k-1}$. The number of iterations required to reduce the search space to a single surviving configuration is given by $K = \log_\eta N$. Thus, the total computational cost incurred by the algorithm satisfies

$$\mathcal{C}_{\text{SH}} = \sum_{k=0}^{K} N_k B_k = \sum_{k=0}^{K} \frac{N}{\eta^k} \cdot \eta^k B_0 = O(B \log_\eta N) \tag{15.387}$$

Compared to brute-force grid search, which incurs an evaluation cost of $\mathcal{C}_{\text{grid}} = NB$, this result demonstrates that SH achieves an **exponential reduction** in computational complexity while maintaining high fidelity in identifying near-optimal hyperparameter configurations. A key probabilistic aspect of SH is its ability to retain at least one optimal configuration with high probability. Let λ^* denote an optimal configuration in Λ, and let $f_k(\lambda)$ represent the performance metric (e.g., validation accuracy) evaluated at iteration k. Assuming $f_k(\lambda)$ follows a sub-Gaussian distribution, the probability that λ^* survives elimination at each iteration satisfies

$$P_k = P\left(f_k(\lambda^*) \geq f_k(\lambda) \text{ for surviving } \lambda\right) \tag{15.388}$$

Applying Chernoff bounds, the probability of discarding λ^* at any given iteration is at most $\frac{1}{\eta^k}$, leading to a final retention probability of

$$P_{\text{final}} = 1 - \frac{1}{\eta^{\log_\eta N}} \tag{15.389}$$

As $N \to \infty$, the term $\frac{1}{\eta^{\log_\eta N}}$ asymptotically vanishes, ensuring that SH converges to an optimal configuration with probability approaching unity. The **asymptotic convergence rate** of SH is given by

$$O\left(\frac{\log N}{N}\right) \tag{15.390}$$

which significantly outperforms naive random search while being slightly suboptimal compared to adaptive bandit-based methods such as Hyperband. Hyperband extends SH by employing multiple independent SH runs with varying initial budget allocations, thereby balancing **exploration** (many configurations trained briefly) and **exploitation** (few configurations trained extensively). The expected number of evaluations required by Hyperband satisfies

$$\mathbb{E}[\text{evaluations}] = O\left(\frac{B \log N}{\log \eta}\right) \tag{15.391}$$

which achieves **sublinear dependence** on N and further enhances computational efficiency. Compared to traditional SH, Hyperband is **more robust** to hyperparameter configurations with delayed performance gains, making it particularly effective for deep learning applications. Despite its computational advantages, SH has several practical limitations. The choice of the reduction factor η influences the algorithm's efficiency; larger values accelerate pruning but increase the risk of discarding promising configurations prematurely. Additionally, SH assumes that partial evaluations of configurations provide an unbiased estimate of their final performance, which may not hold for all machine learning models, particularly those with complex training dynamics. Finally, for small computational budgets B, SH may allocate insufficient resources to any configuration, leading to suboptimal tuning outcomes.

In conclusion, Successive Halving provides a mathematically principled approach to hyperparameter tuning by leveraging **sequential resource allocation** and **early stopping strategies** to reduce computational costs. Its theoretical guarantees ensure that near-optimal configurations are retained with high probability while significantly improving the sample complexity compared to exhaustive search. When coupled with adaptive methods such as Hyperband, SH serves as a **cornerstone of modern hyperparameter optimization**, enabling efficient tuning of high-dimensional models across diverse machine learning applications.

15.4.10 Reinforcement Learning (RL)

Literature Review: Dong et. al. (2019) [519] presented a meta-learning framework where an RL agent learns to optimize hyperparameters across multiple tasks. The authors propose a policy gradient method to train the agent, which generalizes well to unseen optimization problems. The work highlights the transferability of RL-based hyperparameter tuning across different domains. Rijsdijk et. al. (2021) [520] focused on using RL to tune hyperparameters in deep learning models, particularly for neural networks. It introduces a novel RL algorithm that leverages Bayesian optimization as a baseline to guide the search process. The authors demonstrate significant improvements in model performance on benchmark datasets like CIFAR-10 and ImageNet. While not exclusively focused on RL, this work by Snoek et. al. (2012) [489] laid the groundwork for using sequential decision-making in hyperparameter optimization. It introduces Gaussian Process-based Bayesian Optimization, which is often combined with RL techniques. The paper provides a rigorous theoretical framework and practical insights for tuning hyperparameters efficiently. Jaderberg et. al. (2017) [504] proposed a hybrid approach combining RL and evolutionary strategies for hyperparameter tuning. It introduces Population-Based Training (PBT), where a population of models is trained in parallel, and RL is used to adapt hyperparameters dynamically. The method achieves state-of-the-art results in deep reinforcement learning tasks. Jaafra et. al. (2018) [521] explored the use of neural networks as RL agents to optimize hyperparameters. The authors propose a neural architecture search (NAS) framework where the RL agent learns to generate and evaluate hyperparameter configurations. The paper demonstrates the scalability of RL-based methods for large-scale hyperparameter optimization. Afshar and Zhang (2022) [522] introduced a practical RL framework for hyperparameter tuning in machine learning pipelines. It uses a tree-structured Parzen estimator (TPE) to guide the RL agent, enabling efficient exploration of the hyperparameter space. The authors provide empirical evidence of the method's superiority over traditional approaches. Wu et. al. (2020) [523] proposed a model-based RL approach for hyperparameter tuning, where a surrogate model is used to approximate the performance of different hyperparameter configurations. The method reduces the number of evaluations required to find optimal hyperparameters, making it highly efficient for large-scale applications. Iranfar et. al. (2021) [524] focused on using deep RL algorithms, such as Deep Q-Networks (DQN), to optimize hyperparameters in neural networks. The authors demonstrate how deep RL can handle high-dimensional hyperparameter spaces and achieve competitive results on tasks like image classification and natural language processing. While not exclusively about RL, this survey by He et al. (2021) [525] provides a comprehensive overview of automated machine learning (AutoML) techniques, including RL-based hyperparameter tuning. It discusses the strengths and limitations of RL in the context of AutoML and provides a roadmap for future research in the field.

The hyperparameter tuning problem can be rigorously formulated as a stochastic optimization problem:

$$\theta^* = \arg\max_{\theta \in \Theta} \mathbb{E}_{D_{\text{val}}} \left[P(M(\theta); D_{\text{val}}) \right] \tag{15.392}$$

where $\theta \in \Theta$ is the vector of hyperparameters, with Θ being the feasible hyperparameter space, $M(\theta)$ is the machine learning model parameterized by θ, D_{val} is the validation dataset, drawn from a data distribution D, $P(M(\theta); D_{\text{val}})$ is the performance metric (e.g., validation accuracy, negative loss) of the model $M(\theta)$ on D_{val}. This formulation emphasizes that the goal is to optimize the expected performance of the model over the distribution of validation datasets. Let's cast Reinforcement Learning as a Markov Decision Process (MDP). The problem is cast as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ):

- **State Space** (S): The state $s_t \in S$ encodes the current hyperparameter configuration θ_t, the history of performance metrics, and any other relevant information (e.g., computational resources used).

- **Action Space** (A): The action $a_t \in A$ represents a perturbation to the hyperparameters, such that:
$$\theta_{t+1} = \theta_t + a_t. \tag{15.393}$$

- **Transition Dynamics** (P): The transition probability $P(s_{t+1} \mid s_t, a_t)$ describes the stochastic evolution of the state. This includes the effect of training the model $M(\theta_t)$ and evaluating it on D_{val}.

- **Reward Function** (R): The reward $r_t = R(s_t, a_t, s_{t+1})$ quantifies the improvement in model performance, e.g.,
$$r_t = P(M(\theta_{t+1}); D_{\text{val}}) - P(M(\theta_t); D_{\text{val}}). \tag{15.394}$$

- **Discount Factor** (γ): The discount factor $\gamma \in [0, 1]$ balances immediate and future rewards.

The objective is to find a policy $\pi : S \to A$ that maximizes the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{15.395}$$

Let's do Policy Optimization via Stochastic Gradient Ascent, the policy π_ϕ is parameterized by ϕ and optimized using stochastic gradient ascent. The gradient of the expected return $J(\pi_\phi)$ with respect to ϕ is given by the policy gradient theorem:

$$\nabla_\phi J(\pi_\phi) = \mathbb{E}_{\pi_\phi} \left[\nabla_\phi \log \pi_\phi(a_t \mid s_t) Q^\pi(s_t, a_t) \right] \tag{15.396}$$

where $Q^\pi(s_t, a_t)$ is the action-value function, representing the expected return of taking action a_t in state s_t and following policy π thereafter:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k \mid s_t, a_t \right]. \tag{15.397}$$

$\nabla_\phi \log \pi_\phi(a_t \mid s_t)$ is the score function, which measures the sensitivity of the policy to changes in ϕ. To estimate $Q^\pi(s_t, a_t)$, a parameterized value function $Q_w(s_t, a_t)$ is used, where w are the parameters. The value function is optimized by minimizing the mean squared Bellman error:

$$L(w) = \mathbb{E}_{\pi_\phi}\left[(Q_w(s_t, a_t) - (r_t + \gamma Q_w(s_{t+1}, a_{t+1})))^2\right]. \tag{15.398}$$

This is typically solved using stochastic gradient descent:

$$w \leftarrow w - \alpha_w \nabla_w L(w) \tag{15.399}$$

where α_w is the learning rate. We can do exploration via Entropy Regularization. To encourage exploration, an entropy regularization term is added to the policy objective:

$$J_{\text{reg}}(\pi_\phi) = J(\pi_\phi) + \lambda H(\pi_\phi), \tag{15.400}$$

where $H(\pi_\phi)$ is the entropy of the policy:

$$H(\pi_\phi) = \mathbb{E}_{s\sim d^\pi, a\sim\pi}[-\log \pi_\phi(a \mid s)]. \tag{15.401}$$

The entropy term ensures that the policy remains stochastic, thereby facilitating better exploration of the hyperparameter space. Modern RL algorithms for hyperparameter tuning often use advanced policy optimization techniques, such as Proximal Policy Optimization (PPO)

$$L_{\text{CLIP}}(\phi) = \mathbb{E}_t\left[\min\left(\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}A_t, \left(\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A_t\right)\right] \tag{15.402}$$

where the advantage function is defined as:

$$A_t = Q_w(s_t, a_t) - V_w(s_t). \tag{15.403}$$

Trust Region Policy Optimization (TRPO) is

$$\max_\phi \quad \mathbb{E}_t\left[\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}A_t\right] \tag{15.404}$$

$$\text{subject to} \quad \mathbb{E}_t\left[\text{KL}\left(\pi_{\phi_{\text{old}}}(\cdot|s_t)\|\pi_\phi(\cdot|s_t)\right)\right] \leq \delta, \tag{15.405}$$

where KL is the Kullback-Leibler divergence. There are some Theoretical Convergence Guarantees, under certain conditions, RL-based hyperparameter tuning algorithms converge to the optimal policy π^*. Key assumptions include. The MDP satisfies the Bellman optimality principle:

$$Q^*(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \mid s_t, a_t\right]. \tag{15.406}$$

The policy and value function are Lipschitz continuous with respect to their parameters. The learning rates α_ϕ and α_w satisfy the Robbins-Monro conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{15.407}$$

There are some Practical Implementation and Scalability issues. To scale RL-based hyperparameter tuning to high-dimensional spaces, techniques such as:

- **Neural Network Function Approximation:** Use deep neural networks to parameterize the policy π_ϕ and value function Q_w.
- **Parallelization:** Distribute the evaluation of hyperparameter configurations across multiple workers.

- **Early Stopping:** Use techniques like Hyperband to terminate poorly performing configurations early.

We should rigorously analyze the exploration-exploitation tradeoff using multi-armed bandit theory and regret minimization. The cumulative regret $R(T)$ after T steps is defined as:

$$R(T) = \sum_{t=1}^{T} \left(P(M(\theta^*); D_{\text{val}}) - P(M(\theta_t); D_{\text{val}}) \right). \tag{15.408}$$

Algorithms like Upper Confidence Bound (UCB) and Thompson Sampling provide theoretical guarantees on the regret, e.g.,

$$R(T) = O(\sqrt{T}). \tag{15.409}$$

In summary, hyperparameter tuning using reinforcement learning is a mathematically rigorous process that involves first formulating the problem as a stochastic optimization problem within an MDP framework and then Optimizing the policy using advanced gradient-based methods and value function approximation. We then balance exploration and exploitation using entropy regularization and regret minimization and then ensure theoretical convergence and scalability through careful algorithm design and analysis.

15.4.11 Meta-Learning

Literature Review: Gomaa et. al. (2024) [526] introduced SML-AutoML, a novel meta-learning-based automated machine learning (AutoML) framework. It addresses the challenge of model selection and hyperparameter optimization by learning from past experiences. The framework leverages meta-learning to dynamically select the best model architecture and hyperparameters based on historical performance. This research is significant in making AutoML more efficient and adaptable to different datasets. Khan et. al. (2025) [527] explored federated learning where multiple decentralized models collaborate. It proposes a consensus-driven hyperparameter tuning approach using meta-learning to optimize models across nodes. This study is crucial for ensuring model convergence in non-IID (non-independent and identically distributed) data environments, where traditional hyperparameter optimization methods often fail. Morrison and Ma (2025) [528] focused on meta-optimization for improving machine learning optimizers. The study evaluates various optimization algorithms, demonstrating that meta-learning can fine-tune optimizer hyperparameters to improve model efficiency, particularly in nanophotonic inverse design tasks. This approach is applicable in physics-driven AI models that require precise parameter tuning. Berdyshev et. al. (2025) [529] presented EEG-Reptile, a meta-learning framework for brain-computer interfaces (BCI) that tunes hyperparameters dynamically during learning. The study introduces a Reptile-based meta-learning approach that enables fast adaptation of models to individual brain signal patterns, making AI-powered BCI systems more personalized and efficient. Pratellesi (2025) [530] applied meta-learning to biomedical classification problems, specifically in flow cytometry cell analysis. The paper demonstrates that meta-learning can optimize hyperparameter selection for imbalanced biomedical datasets, improving classification accuracy while reducing computational costs. Garcia et. al. (2022) [531] introduced a meta-learned Bayesian hyperparameter search technique for metabolite annotation. It highlights how meta-learning can improve molecular property prediction by selecting optimal descriptors and hyperparameters for chemical space exploration. Deng et. al. (2024) [532] introduced a surrogate modeling approach that leverages meta-learning for efficient hyperparameter search. The proposed method significantly reduces the computational cost of hyperparameter tuning while maintaining high performance. The study is particularly useful for computationally expensive AI models like deep neural networks. Jae et. al. (2024) [533] integrated reinforcement learning with meta-learning to optimize hyperparameters for quantum state learning. It demonstrates how reinforcement learning agents can dynamically adjust hyperparameters, improving black-box optimization methods for quantum computing applications.

Upadhyay et. al. (2025) [534] investigated meta-learning-based sparsity optimization in multi-task networks. By learning the optimal sparsity structure and hyperparameters, this approach enhances memory efficiency and computational scalability for large-scale deep learning applications. Paul et. al. (2025) [535] provided a comprehensive theoretical and practical overview of meta-learning for neural network design. It discusses how meta-learning can automate hyperparameter tuning, improve transfer learning strategies, and enhance architecture search.

The selection of hyperparameters, denoted by θ, plays a pivotal role in determining the model's performance. This selection process, when viewed through the lens of optimization theory, can be formulated as a global optimization problem where the goal is to minimize the expected loss over a distribution of datasets $p(\mathcal{D})$:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})} \left[\mathcal{L}(f_\theta(\mathcal{D}))\right] \tag{15.410}$$

Here, $\mathcal{D}$ denotes the dataset, and $\mathcal{L}$ is the loss function used to measure the quality of the model. The challenge arises because the hyperparameters θ are fixed before training begins, unlike the model parameters that are learned via optimization techniques such as gradient descent. This problem becomes computationally intractable when θ is high-dimensional or when traditional grid and random search methods are employed. Meta-learning, often referred to as "learning to learn," provides a sophisticated framework to address hyperparameter tuning. The key objective in meta-learning is to develop a meta-model that can efficiently adapt to new tasks with minimal data. Mathematically, consider a set of tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$, where each task T_i consists of a dataset $\mathcal{D}_i$ and a corresponding loss function $\mathcal{L}_i$. The meta-learning framework aims to find meta-parameters ϕ that minimize the expected loss across tasks:

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{T\sim p(T)} \left[\mathcal{L}(f_{\theta_T}, T)\right] \tag{15.411}$$

Here, $\theta_T = h(\mathcal{D}_T, \phi)$ is a task-specific hyperparameter derived from the meta-parameters ϕ. The inner optimization problem, which corresponds to the task-specific optimization of θ_T, is given by:

$$\theta_T^* = \arg\min_{\theta} \mathcal{L}_T(f_\theta, \mathcal{D}_T) \tag{15.412}$$

Meanwhile, the outer optimization problem concerns learning ϕ, the meta-parameters, from multiple tasks:

$$\phi^* = \arg\min_{\phi} \sum_{T_i\in\mathcal{T}} \mathcal{L}_T(f_{h(\mathcal{D}_T,\phi)}, \mathcal{D}_T) \tag{15.413}$$

This nested optimization structure, wherein the inner optimization problem is task-specific and the outer optimization problem is meta-specific, requires careful treatment via gradient-based methods and implicit differentiation. The meta-learning process can be understood as a bi-level optimization problem. To analyze this, we first consider the **inner optimization**, which optimizes the task-specific hyperparameters θ for each task T_i. This is given by:

$$\theta_i^* = \arg\min_{\theta} \mathcal{L}_i(f_\theta, \mathcal{D}_i) \tag{15.414}$$

For each task, the hyperparameter θ is chosen to minimize the corresponding task-specific loss. The **outer optimization** then aims to find the optimal meta-parameters ϕ across tasks. The outer objective can be written as:

$$\phi^* = \arg\min_{\phi} \sum_{i=1}^{N} \mathcal{L}_i(f_{h(\mathcal{D}_i,\phi)}, \mathcal{D}_i) \tag{15.415}$$

Since the task-specific loss $\mathcal{L}_i$ depends on θ_i^*, which in turn depends on ϕ, we require the application of **implicit differentiation**. By applying the chain rule, we obtain the gradient of the outer objective with respect to ϕ:

$$\nabla_\phi \mathcal{L}_i(f_{\theta_i^*}, \mathcal{D}_i) = \nabla_{\theta_i^*} \mathcal{L}_i \cdot \frac{\partial \theta_i^*}{\partial \phi} \tag{15.416}$$

The term $\frac{\partial \theta_i^*}{\partial \phi}$ involves the inverse of the Hessian matrix of the loss function with respect to θ, leading to a computationally expensive second-order update rule:

$$\frac{\partial \theta_i^*}{\partial \phi} \approx -\left(\nabla_{\theta_i}^2 \mathcal{L}_i\right)^{-1} \nabla_{\theta_i} h(\mathcal{D}_i, \phi) \tag{15.417}$$

This analysis demonstrates the intricate dependencies between the task-specific hyperparameters and the meta-parameters, requiring sophisticated optimization strategies for practical use. Gradient-Based Meta-Learning (e.g., Model-Agnostic Meta-Learning or MAML) seeks to find an optimal initialization θ_0 for the hyperparameters that can be adapted to new tasks with a small number of gradient steps. For a single task T_i, the hyperparameters are adapted as follows:

$$\theta_i' = \theta_0 - \alpha \nabla_\theta \mathcal{L}_i(f_{\theta_0}, \mathcal{D}_i) \tag{15.418}$$

Here, α is the learning rate for task-specific updates. The goal is to optimize θ_0 such that, after a few gradient steps, the model performs well on any task T_i. The meta-objective is given by:

$$\min_{\theta_0} \sum_{i=1}^{N} \mathcal{L}_i(f_{\theta_i'}, \mathcal{D}_i) \tag{15.419}$$

Taking the gradient of the meta-objective with respect to θ_0, we obtain:

$$\nabla_{\theta_0} \left(\sum_{i=1}^{N} \mathcal{L}_i(f_{\theta_i'}, \mathcal{D}_i) \right) = \sum_{i=1}^{N} \nabla_{\theta_i'} \mathcal{L}_i \cdot \frac{\partial \theta_i'}{\partial \theta_0} \tag{15.420}$$

Here, $\frac{\partial \theta_i'}{\partial \theta_0}$ involves a term that accounts for the task-specific gradients, leading to an efficient update rule. The application of second-order optimization methods such as **Hessian-free optimization** or **L-BFGS** is critical in achieving computational efficiency. Bayesian meta-learning models the uncertainty over hyperparameters using probabilistic methods, with a primary focus on uncertainty propagation. In this approach, we assume that hyperparameters follow a distribution:

$$\theta \sim p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{15.421}$$

A popular choice is the **Gaussian Process (GP)**, which provides a distribution over functions. For hyperparameter optimization, we define a prior over the hyperparameters as:

$$\theta \sim \mathcal{GP}(\mu, K) \tag{15.422}$$

where $K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2l^2}\right)$ is the RBF kernel, and l is the length scale parameter. The posterior distribution over θ given the observed data is:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{15.423}$$

Using this posterior, we define an acquisition function such as **Expected Improvement (EI)**:

$$\text{EI}(\theta) = \mathbb{E}\left[\max(0, f(\theta) - f^*)\right] \tag{15.424}$$

which helps guide the optimization of θ by balancing exploration and exploitation. The computational challenges in this approach are mitigated by using **sparse Gaussian Processes** or **variational inference** methods, which approximate the posterior more efficiently. In conclusion, Meta-learning offers a mathematically rigorous framework for hyperparameter tuning, leveraging advanced optimization techniques and probabilistic models to adapt to new tasks efficiently. The bi-level optimization problem, second-order derivatives, and Bayesian frameworks provide both theoretical depth and practical utility. These sophisticated methods represent a powerful toolkit for hyperparameter optimization in complex machine learning systems.

16 Convolution Neural Networks

Literature Review: Goodfellow et. al. (2016) [112] wrote one of the most foundational textbooks on deep learning, covering CNNs in depth. It introduces theoretical principles, including convolutions, backpropagation, and optimization methods. The book also discusses applications of CNNs in image processing and beyond. LeCun et. al. (2015) [117] provides a historical overview of CNNs and deep learning. LeCun, one of the inventors of CNNs, explains why convolutions help in image recognition and discusses their applications in vision, speech, and reinforcement learning. Krizhevsky et. al. (2012) [146] and Krizhevsky et. al. (2017) [147] introduced AlexNet, the first modern deep CNN, which won the 2012 ImageNet Challenge. It demonstrated that deep CNNs can achieve unprecedented accuracy in image classification tasks, paving the way for deep learning's dominance. Simonyan and Zisserman (2015) [148] introduced VGGNet, which demonstrated that increasing network depth using small 3x3 convolutions can improve performance. It also provided insights into layer design choices and their effects on accuracy. He et. al. (2016) [149] introduced ResNet, which solved the vanishing gradient problem in deep networks by using skip connections. This revolutionized CNN design by allowing models as deep as 1000 layers to be trained efficiently. Cohen and Welling (2016) [150] extended CNNs using group theory, enabling equivariant feature learning. This improved CNN robustness to rotations and translations, making them more efficient in symmetry-based tasks. Zeiler and Fergus (2014) [151] introduced deconvolution techniques to visualize CNN feature maps, making it easier to interpret and debug CNNs. It showed how different layers detect patterns, textures, and objects. Liu et.al. (2021) [152] introduced Vision Transformers (ViTs) that outperform CNNs in some vision tasks. This paper discusses the limitations of CNNs and how transformers can be hybridized with CNN architectures. Lin et.al. (2013) [153] introduced the 1x1 convolution, which improved feature learning efficiency. This concept became a key component of modern CNN architectures such as ResNet and MobileNet. Rumelhart et. al. (1986) [154] formalized backpropagation, the training method used for CNNs. Without this discovery, CNNs and deep learning would not exist today.

16.1 Key Concepts

A **Convolutional Neural Network (CNN)** is a deep learning model primarily used for analyzing grid-like data, such as images, video, and time-series data with spatial or temporal dependencies. The fundamental operation of CNNs is the **convolution** operation, which is employed to extract local patterns from the input data. The input to a CNN is generally represented as a tensor $\mathbf{I} \in \mathbb{R}^{H \times W \times C}$, where H is the height, W is the width, and C is the number of channels (for RGB images, $C = 3$).

At the core of a CNN is the convolutional layer, where the input image $\mathbf{I}$ is convolved with a set of filters or kernels $\mathbf{K} \in \mathbb{R}^{f_h \times f_w \times C}$, where f_h and f_w are the height and width of the filter, respectively. The filter $\mathbf{K}$ slides across the input image $\mathbf{I}$, and the result of this convolution is a set of feature maps that are indicative of certain local patterns in the image. The element-wise convolution at location (i, j) of the feature map is given by:

$$\mathbf{I} * \mathbf{K} = \sum_{p=1}^{f_h} \sum_{q=1}^{f_w} \sum_{r=1}^{C} \mathbf{I}_{i+p-1,j+q-1,r} \cdot \mathbf{K}_{p,q,r} \tag{16.1}$$

where $\mathbf{I}_{i+p-1,j+q-1,r}$ denotes the value of the r-th channel of the input image at position $(i+p-1, j+q-1)$, and $\mathbf{K}_{p,q,r}$ is the corresponding filter value at (p, q, r). This operation is done for each location (i, j) of the output feature map. The resulting feature map $\mathbf{F}$ has spatial dimensions $H' \times W'$, where:

$$H' = \left\lfloor \frac{H + 2p - f_h}{s} \right\rfloor + 1, \quad W' = \left\lfloor \frac{W + 2p - f_w}{s} \right\rfloor + 1 \tag{16.2}$$

where p is the padding, and s is the stride of the filter during its sliding motion. The convolution operation provides a translation-invariant representation of the input image, as each filter detects patterns across the entire image. After this convolution, a non-linear activation function, typically the **Rectified Linear Unit (ReLU)**, is applied to introduce non-linearity into the network and ensure it can model complex patterns. The ReLU activation function operates element-wise and is given by:

$$\text{ReLU}(x) = \max(0, x) \tag{16.3}$$

Thus, for each feature map $\mathbf{F}$, the output after ReLU is:

$$\mathbf{F}'_{i,j,k} = \max(0, \mathbf{F}_{i,j,k}) \tag{16.4}$$

This ensures that negative values in the feature map are discarded, which helps with the sparse representation of activations, mitigating the vanishing gradient problem in deeper layers. In CNNs, pooling operations follow the convolution and activation layers. Pooling serves to reduce the spatial dimensions of the feature maps, thus decreasing computational complexity and making the representation more invariant to translations. **Max pooling**, which is the most common form, selects the maximum value within a specified window size $p_h \times p_w$. Given an input feature map $\mathbf{F} \in \mathbb{R}^{H' \times W' \times K}$, max pooling operates as follows:

$$\mathbf{P}_{i,j,k} = \max\left(\mathbf{F}_{i,j,k}, \mathbf{F}_{i+1,j,k}, \mathbf{F}_{i,j+1,k}, \mathbf{F}_{i+1,j+1,k}\right) \tag{16.5}$$

where $\mathbf{P}$ is the pooled feature map. This pooling operation effectively reduces the spatial dimensions of each feature map, resulting in an output $\mathbf{P} \in \mathbb{R}^{H'' \times W'' \times K}$, where:

$$H'' = \left\lfloor \frac{H'}{p_h} \right\rfloor, \quad W'' = \left\lfloor \frac{W'}{p_w} \right\rfloor \tag{16.6}$$

Max pooling introduces an element of robustness by capturing only the strongest features within the local regions, discarding irrelevant information, and ensuring that the network is invariant to small translations. The CNN architecture typically contains multiple convolutional layers followed by pooling layers. After these operations, the feature maps are flattened into a one-dimensional vector and passed into one or more **fully connected (dense) layers**. A fully connected layer computes a linear transformation of the form:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{16.7}$$

where $\mathbf{a}^{(l-1)}$ is the input to the layer, $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias vector. The output of this linear transformation is then passed through a non-linear activation function, such as ReLU or softmax for classification tasks. For classification, the **softmax** function is often applied to convert the output into a probability distribution:

$$\mathbf{y}_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} \tag{16.8}$$

where C is the number of output classes, and $\mathbf{y}_i$ is the probability of the i-th class. The softmax function ensures that the output probabilities sum to 1, providing a valid classification output. The CNN is trained using **backpropagation**, which computes the gradients of the loss function $\mathcal{L}$ with respect to the network's parameters (i.e., weights and biases). Backpropagation uses the **chain rule** to propagate the error gradients through each layer. The gradients with respect to the convolutional filters $\mathbf{K}$ are computed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{K}} = \frac{\partial \mathcal{L}}{\partial \mathbf{F}} * \mathbf{I} \tag{16.9}$$

where $*$ denotes the convolution operation. Similarly, the gradients for the fully connected layers are computed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \tag{16.10}$$

Once the gradients are computed, the weights are updated using an optimization algorithm like **gradient descent**:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} \tag{16.11}$$

where η is the learning rate. This optimization ensures that the network's parameters are adjusted in the direction of the negative gradient, minimizing the loss function and thereby improving the performance of the CNN. Regularization techniques are commonly applied to avoid overfitting. **Dropout**, for instance, randomly deactivates a subset of neurons during training, preventing the network from becoming too reliant on any specific feature and promoting better generalization. The dropout operation at a given layer l with dropout rate p is defined as:

$$\mathbf{a}^{(l)} \sim \text{Dropout}(\mathbf{a}^{(l)}, p) \tag{16.12}$$

where the activations $\mathbf{a}^{(l)}$ are randomly set to zero with probability p, and the remaining activations are scaled by $\frac{1}{1-p}$. Another regularization technique is **batch normalization**, which normalizes the inputs of each layer to have zero mean and unit variance, thus improving training speed and stability. Mathematically, batch normalization is defined as:

$$\hat{x} = \frac{x - \mu_B}{\sigma_B}, \quad \mathbf{y} = \gamma \hat{x} + \beta \tag{16.13}$$

where μ_B and σ_B are the mean and standard deviation of the batch, and γ and β are learned scaling and shifting parameters.

In conclusion, the mathematical backbone of a **Convolutional Neural Network (CNN)** relies heavily on the convolution operation, non-linear activations, pooling, and fully connected transformations. The convolutional layers extract hierarchical features by applying filters to the input data, while pooling reduces the spatial dimensions and introduces invariance to translations. The fully connected layers aggregate these features for classification or regression tasks. The network is trained using backpropagation and optimization techniques such as gradient descent. Regularization methods like dropout and batch normalization are used to improve generalization and training efficiency. The mathematical formalism behind CNNs is essential for understanding their power in various machine learning tasks, particularly in computer vision.

16.2 Applications in Image Processing

16.2.1 Image Classification

Literature Review: Thiriveedhi et. al. (2025) [185] presented a novel CNN-based architecture for diagnosing Acute Lymphoblastic Leukemia (ALL), integrating explainable AI (XAI) techniques.

The proposed model outperforms traditional CNNs by providing human-interpretable insights into medical image classification. The research highlights how CNNs can be effectively applied to medical imaging with enhanced transparency. Ramos-Briceño et. al. (2025) [186] demonstrated the superior classification accuracy of CNNs in malaria parasite detection. The research uses deep CNNs to classify malaria species in blood samples and achieves state-of-the-art performance. The paper provides valuable insights into CNN-based image classification for biomedical applications. Espino-Salinas et. al. (2025) [187] applied CNNs to mental health diagnostics by classifying motion activity patterns as images. The paper explores the novel application of CNNs beyond traditional image classification by transforming time-series data into visual representations and utilizing CNNs to detect psychiatric disorders. Ran et. al. (2025) [188] introduced a CNN-based hyperspectral imaging method for early diagnosis of pancreatic neuroendocrine tumors. The paper highlights CNNs' ability to process multispectral data for complex medical imaging tasks, further expanding their utility in pathology and cancer detection. Araujo et. al. (2025) [189] demonstrated how CNNs can be employed in industrial monitoring and predictive maintenance. The research introduces an innovative CNN-based approach for detecting faults in ZnO surge arresters using thermal imaging, proving CNNs' robustness in non-destructive testing applications. Sari et. al. (2025) [190] applied CNNs to cultural heritage preservation, specifically Batik pattern classification. The study showcases CNNs' adaptability in fine-grained image classification and highlights the importance of deep learning in automated textile pattern recognition. Wang et. al. (2025) [191] proposed CF-WIAD, a novel semi-supervised learning method that leverages CNNs for skin lesion classification. The research demonstrates how CNNs can be used to effectively classify dermatological images, particularly in low-data environments, which is a key challenge in medical AI. Cai et. al. (2025) [192] introduced DFNet, a CNN-based residual network that improves feature extraction by incorporating differential features. The study highlights CNNs' role in advanced feature engineering, which is crucial for applications such as facial recognition and object classification. Vishwakarma and Deshmukh (2025) [193] presented CNNM-FDI, a CNN-based fire detection model that enhances real-time safety monitoring. The study explores CNNs' application in environmental monitoring, emphasizing fast-response classification models for early disaster prevention. Ranjan et. al. (2025) [194] merged CNNs, Autoencoders, GANs, and Zero-Shot Learning to improve hyperspectral image classification. The research underscores how CNNs can be augmented with generative models to enhance classification in limited-label datasets, a crucial area in remote sensing applications.

The process of image classification in Convolutional Neural Networks (CNNs) involves a sophisticated interplay of linear algebra, calculus, probability theory, and optimization. The primary goal is to map a high-dimensional input image to a specific class label. Let $\mathbf{I} \in \mathbb{R}^{H\times W\times C}$ represent the input image, where H, W, and C are the height, width, and number of channels (usually 3 for RGB images) of the image, respectively. Each pixel of the image can be represented as $\mathbf{I}(i, j, c)$, which denotes the intensity of the c-th channel at pixel position (i, j). The objective of the CNN is to transform this raw input image into a label, typically one of M classes, using a hierarchical feature extraction process that includes convolutions, nonlinearities, pooling, and fully connected layers.

The convolution operation is central to CNNs and forms the basis for the feature extraction process. Let $\mathbf{K} \in \mathbb{R}^{k\times k\times C}$ be a filter (or kernel) with spatial dimensions $k \times k$ and C channels, where k is typically a small odd integer, such as 3 or 5. The filter $\mathbf{K}$ is convolved with the input image $\mathbf{I}$ to produce a feature map $\mathbf{S} \in \mathbb{R}^{(H-k+1)\times(W-k+1)\times F}$, where F is the number of filters used in the convolution. For a given spatial position (i, j) in the feature map, the convolution operation is defined as:

$$\mathbf{S}_{i,j,f} = \sum_{m=0}^{k-1}\sum_{n=0}^{k-1}\sum_{c=0}^{C-1} \mathbf{I}(i+m, j+n, c) \cdot \mathbf{K}_{m,n,c,f} \tag{16.14}$$

where $\mathbf{S}_{i,j,f}$ represents the value at position (i, j) in the feature map corresponding to the f-th filter. This operation computes a weighted sum of pixel values in the receptive field of size $k \times k \times C$ around pixel (i, j), where the weights are given by the filter values. The result is a new feature map that captures local patterns such as edges or textures in the image. This local feature extraction is performed for each position (i, j) across the entire image, producing a set of feature maps for each filter. To introduce non-linearity into the network and allow it to model complex functions, the feature map $\mathbf{S}$ is passed through a non-linear activation function, typically the Rectified Linear Unit (ReLU), which is defined element-wise as:

$$\sigma(x) = \max(0, x) \tag{16.15}$$

This activation function outputs 0 for negative values and passes positive values unchanged, ensuring that the network can learn complex, non-linear relationships. The output of the activation function for the feature map is denoted as $\mathbf{S}^+$, where each element of $\mathbf{S}^+$ is computed as:

$$\mathbf{S}^+_{i,j,f} = \max(0, \mathbf{S}_{i,j,f}) \tag{16.16}$$

This element-wise operation enhances the network's ability to capture and represent complex patterns, thereby aiding in the learning process. After the convolution and activation, the feature map is downsampled using a pooling operation. The most common form of pooling is max pooling, which selects the maximum value in a local region of the feature map. Given a pooling window of size $p \times p$ and stride s, the max pooling operation for the feature map $\mathbf{S}^+$ is given by:

$$\mathbf{P}_{i,j,f} = \max_{(u,v) \in p \times p} \mathbf{S}^+_{i+u,j+v,f} \tag{16.17}$$

where $\mathbf{P}$ represents the pooled feature map. This operation reduces the spatial dimensions of the feature map by a factor of p, while preserving the most important features in each region. Pooling serves several purposes, including dimensionality reduction, translation invariance, and noise reduction. It also helps prevent overfitting by limiting the number of parameters and computations in the network.

Once the feature maps are obtained through convolution, activation, and pooling, they are flattened into a one-dimensional vector $\mathbf{F} \in \mathbb{R}^N$, where N is the total number of elements in the pooled feature map. The flattened vector $\mathbf{F}$ is then fed into one or more fully connected layers. These layers perform linear transformations of the input, which are essentially weighted sums of the inputs, followed by the addition of a bias term. The output of a fully connected layer can be expressed as:

$$\mathbf{O} = \mathbf{W} \cdot \mathbf{F} + \mathbf{b} \tag{16.18}$$

where $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^M$ is the bias vector, and $\mathbf{O} \in \mathbb{R}^M$ is the raw output or logit for each of the M classes. The fully connected layer computes a set of logits for the classes based on the learned features from the convolutional and pooling layers. To convert the logits into class probabilities, a **softmax** function is applied. The softmax function is a generalization of the logistic function to multiple classes and transforms the logits into a probability distribution. The probability of class k is given by:

$$P(y = k \mid \mathbf{O}) = \frac{e^{O_k}}{\sum_{k=1}^{M} e^{O_k}} \tag{16.19}$$

where O_k is the logit corresponding to class k, and the denominator ensures that the sum of probabilities across all classes equals 1. The class label with the highest probability is selected as the final prediction:

$$y = \arg\max_k P(y = k \mid \mathbf{O}) \tag{16.20}$$

The prediction is made based on the computed class probabilities, and the network aims to minimize the discrepancy between the predicted probabilities and the true labels during training. To optimize the network's parameters, we minimize a **loss function** that measures the difference between the predicted probabilities and the actual labels. The **cross-entropy loss** is commonly used in classification tasks and is defined as:

$$\mathcal{L} = -\sum_{k=1}^{M} y_k \log P(y = k \mid \mathbf{O}) \tag{16.21}$$

where y_k is the true label in one-hot encoding, and $P(y = k \mid \mathbf{O})$ is the predicted probability for class k. The goal of training is to minimize this loss function, which corresponds to maximizing the likelihood of the correct class under the predicted probability distribution.

The optimization of the network parameters is performed using **gradient descent** and its variants, such as stochastic gradient descent (SGD), which iteratively updates the parameters based on the gradients of the loss function. The gradients are computed using **backpropagation**, a method that applies the chain rule of calculus to compute the partial derivatives of the loss with respect to each parameter. For a fully connected layer, the gradient of the loss with respect to the weights $\mathbf{W}$ is given by:

$$\nabla_{\mathbf{W}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{O}} \cdot \frac{\partial \mathbf{O}}{\partial \mathbf{W}} = \delta \cdot \mathbf{F}^T \tag{16.22}$$

where $\delta = \frac{\partial \mathcal{L}}{\partial \mathbf{O}}$ is the error term (also known as the delta) for the logits, and $\mathbf{F}^T$ is the transpose of the flattened feature vector. The parameters are updated using the following rule:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L} \tag{16.23}$$

where η is the learning rate, controlling the step size of the updates. This process is repeated for each batch of training data until the network converges to a set of parameters that minimize the loss function. Through this complex and iterative process, CNNs are able to learn to classify images by automatically extracting hierarchical features from raw input data. The combination of convolution, activation, pooling, and fully connected layers enables the network to learn increasingly abstract and high-level representations of the input image, ultimately achieving high accuracy in image classification tasks.

16.2.2 Object Detection

Literature Review: Naseer and Jalal (2025) [195] presented a multimodal deep learning framework that integrates RGB-D images for enhanced semantic scene classification. The study leverages a Convolutional Neural Network (CNN)-based object detection model to extract and process features from RGB and depth images, aiming to improve scene recognition accuracy in cluttered and complex environments. By incorporating multimodal inputs, the model effectively addresses the challenges associated with occlusions and background noise, which are common issues in traditional object detection frameworks. The researchers demonstrate how CNNs, when combined with depth-aware semantic information, can significantly enhance object localization and classification performance. Through extensive evaluations, they validate that their framework outperforms conventional single-stream CNNs in various real-world scenarios, making a compelling case for RGB-D integration in deep learning-based object detection systems. Wang and Wang (2025) [196] builds upon the Faster R-CNN object detection framework, introducing a novel improvement that significantly enhances detection accuracy in highly dynamic and complex environments. The study proposes an optimized anchor box generation mechanism, which allows the network to efficiently detect objects of varying scales and aspect ratios, particularly those that are small or heavily occluded. By incorporating a refined region proposal network (RPN), the authors mitigate localization errors and reduce false-positive detections. The paper also explores the impact of feature pyramid networks (FPNs) in hierarchical feature extraction, demonstrating their effectiveness in improving the detection of fine-grained details. The authors conduct an extensive empirical evaluation, comparing their improved Faster R-CNN model against existing object detection architectures, proving its superior performance in terms of precision and recall, particularly for applications involving customized icon generation and user interface design. Ramana et. al. (2025) [197] introduced a Deep Convolutional Graph Neural Network (DCGNN) that integrates Spectral Pyramid Pooling (SPP) and fused keypoint generation to significantly improve 3D object detection performance. The study employs ResNet-50 as the backbone CNN architecture and enhances its feature extraction capability by introducing multi-scale spectral feature aggregation. Through the integration of graph neural networks (GNNs), the model can effectively capture spatial relationships between object components, leading to highly accurate 3D bounding box predictions. The proposed methodology is rigorously evaluated on multiple benchmark datasets, demonstrating its superior ability to handle occlusion, scale variation, and viewpoint changes. Additionally, the paper presents a novel fusion strategy that combines keypoint-based object representation with spectral domain feature embeddings, allowing the model to achieve unparalleled robustness in automated 3D object detection tasks. Shin et. al. (2025) [198] explores the application of deep learning-based object detection in the field of microfluidics and droplet-based bioengineering. The authors utilize YOLOv10n, an advanced CNN-based object detection framework, to develop an automated system for tracking and categorizing double emulsion droplets in high-throughput experimental setups. By fine-tuning the YOLO architecture, the study achieves remarkable improvements in detection sensitivity and classification accuracy, enabling real-time identification of droplet morphology, phase separation dynamics, and stability characteristics. The researchers further introduce an adaptive feature refinement strategy, wherein the CNN model continuously learns from real-time experimental variations, allowing for automated calibration and correction of droplet misclassification. The paper also demonstrates the practical implications of this AI-driven approach in drug delivery systems, encapsulation technologies, and synthetic biology applications. Taca et. al. (2025) [199] provided a comprehensive comparative analysis of multiple CNN-based object detection architectures applied to aphid classification in large-scale agricultural datasets. The researchers evaluate the performance of YOLO, SSD, Faster R-CNN, and EfficientDet, analyzing their trade-offs in terms of accuracy, inference speed, and computational efficiency. Through an extensive experimental setup involving 48,000 annotated images, the authors demonstrate that certain CNN models excel in specific detection scenarios, such as YOLO for real-time aphid localization and Faster R-CNN for high-precision classification. Furthermore, the paper introduces an innovative hybrid ensemble strategy, combining the strengths of multiple CNN architectures to achieve optimal detection performance. The authors validate their findings on real-world agricultural environments, emphasizing the importance of deep learning-driven pest detection in sustainable farming practices. Ulaş et. al. (2025) [200] explored the application of CNN-based object detection in the domain of astronomical time-series analysis, specifically targeting oscillation-like patterns in eclipsing binary light curves. The study systematically evaluates multiple state-of-the-art object detection models, including YOLO, Faster R-CNN, and SSD, to determine their effectiveness in identifying transient light fluctuations that indicate oscillatory behavior in celestial bodies. One of the key contributions of this paper is the introduction of a customized pre-processing pipeline that optimizes raw observational data by removing noise and enhancing feature visibility using wavelet-based signal decomposition techniques. The researchers further implement a hybrid detection mechanism, integrating CNN-based spatial feature extraction with recurrent neural networks (RNNs) to capture both spatial and temporal dependencies within light curve datasets. Extensive validation on large-scale astronomical datasets demonstrates that this approach significantly outperforms traditional statistical methods in detecting oscillatory behavior, paving the way for AI-driven automation in astrophysical event classification. Valensi et. al. (2025) [201] presents an advanced semi-supervised deep learning framework for pleural line detection and segmentation in lung ultrasound (LUS) imaging, leveraging the power of foundation models and CNN-based object detection architectures. The study highlights the shortcomings of conventional fully supervised learning in medical imaging, where annotated datasets are limited and labor-intensive to create. To overcome this challenge, the researchers incorporate a semi-supervised learning strategy, utilizing self-training techniques combined with pseudo-labeling to improve model generalization. The framework employs YOLOv8-based object detection, specifically optimized for medical feature localization, which significantly enhances detection accuracy in cases of low-contrast and high-noise ultrasound images. Furthermore, the study integrates a multi-scale feature extraction strategy, combining convolutional layers with attention mechanisms to ensure precise identification of pleural lines across different imaging conditions. Experimental results demonstrate that this hybrid approach achieves a substantial increase in segmentation accuracy, particularly in detecting subtle abnormalities linked to pneumothorax and pleural effusion, making it a highly valuable tool in clinical diagnostic applications. Arulalan et. al. (2025) [202] proposed an optimized object detection pipeline that integrates a novel convolutional neural network (CNN) architecture, BS2ResNet, with bidirectional LSTM (LTK-Bi-LSTM) for improved spatiotemporal object recognition. Unlike conventional CNN-based object detectors, which focus solely on static spatial features, this study introduces a hybrid deep learning framework that captures both spatial and temporal dependencies. The proposed BS2ResNet model enhances feature extraction by utilizing bottleneck squeeze-and-excitation blocks, which selectively emphasize important spatial information while suppressing redundant feature maps. Additionally, the integration of LTK-Bi-LSTM layers allows the model to effectively capture temporal correlations, making it highly robust for detecting moving objects in dynamic environments. This approach is validated on multiple benchmark datasets, including autonomous driving and video surveillance datasets, where it demonstrates superior performance in handling occlusions, rapid motion, and low-light conditions. The findings indicate that combining deep convolutional networks with sequence-based modeling significantly improves object detection accuracy in complex real-world scenarios, offering critical advancements for applications in intelligent transportation, security, and real-time monitoring. Zhu et. al. (2025) [203] investigated a novel adversarial attack strategy targeting CNN-based object detection models, with a specific focus on binary image segmentation tasks such as salient object detection and camouflage object detection. The paper introduces a high-transferability adversarial attack framework, which generates adversarial perturbations capable of fooling a wide range of deep learning models, including YOLO, Mask R-CNN, and U-Net-based segmentation networks. The researchers employ adversarial example augmentation, where synthetic adversarial patterns are iteratively refined through gradient-based optimization techniques, ensuring that the adversarial attacks remain effective across different architectures and datasets. A particularly important contribution is the introduction of a dual-stage attack pipeline, wherein the model first learns to generate localized, high-impact adversarial noise and then optimizes for cross-model generalization. Extensive experiments demonstrate that this approach significantly degrades detection performance across multiple state-of-the-art models, revealing critical vulnerabilities in current CNN-based object detectors. This research provides valuable insights into deep learning security and underscores the urgent need for robust adversarial defense mechanisms in high-stakes applications such as autonomous systems, medical imaging, and biometric security. Guo et. al. (2025) [204] introduced a deep learning-based agricultural monitoring system, utilizing CNNs for agronomic entity detection and attribute extraction. The research highlights the limitations of traditional rule-based and manual annotation systems in agricultural monitoring, which are prone to errors and inefficiencies. By leveraging CNN-based object detection models, the proposed system enables real-time crop analysis, accurately identifying key agronomic attributes such as plant height, leaf structure, and disease symptoms. A significant innovation in this study is the incorporation of inter-layer feature fusion, wherein multi-scale convolutional features are integrated across different network depths to improve detection robustness under varying lighting and environmental conditions. Additionally, the authors employ a hybrid feature selection mechanism, combining spatial attention networks with spectral domain feature extraction, which enhances the model's ability to distinguish between healthy and diseased crops with high precision. The research is validated through rigorous field trials, demonstrating that CNN-based agronomic monitoring can significantly enhance crop yield predictions, reduce human labor in precision agriculture, and optimize resource allocation in farming operations.

Object detection in Convolutional Neural Networks (CNNs) is a multifaceted computational process that intertwines both classification and localization. It involves detecting objects within an image and predicting their positions via bounding boxes. This task can be mathematically decomposed into the combined problems of classification and regression, both of which are intricately handled by the convolutional layers of a deep neural network. These layers extract hierarchical features at different levels of abstraction, starting from low-level features like edges and corners to high-level semantic concepts such as textures and object parts. These feature maps are then processed by fully connected layers for classification and bounding box regression tasks.

In the mathematical framework, let the input image be represented by a matrix $I \in \mathbb{R}^{H \times W \times C}$, where H, W, and C are the height, width, and number of channels (typically 3 for RGB images). Convolution operations in a CNN serve as the fundamental building blocks to extract spatial hierarchies of features. The convolution operation involves the application of a kernel $K \in \mathbb{R}^{m \times n \times C}$ to the input image, where m and n are the spatial dimensions of the kernel, and C is the number of input channels. The convolution operation is performed by sliding the kernel over the image and computing the element-wise multiplication between the kernel and the image patch, yielding the following equation for the feature map $O(x, y)$:

$$O(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{c=0}^{C-1} I(x+i, y+j, c) \cdot K(i, j, c) \tag{16.24}$$

Here, $O(x, y)$ represents the feature map at the location (x, y), which is generated by applying the kernel K. The sum is taken over the spatial extent of the kernel as it slides over the image. This convolutional operation helps the network capture local patterns in the input image, such as edges, corners, and textures, which are crucial for identifying objects. Once the convolution is performed, a non-linear activation function such as the Rectified Linear Unit (ReLU) is applied to introduce non-linearity into the system. The ReLU activation function is given by:

$$f(x) = \max(0, x) \tag{16.25}$$

This activation function helps the network model complex non-linear relationships between features and is computationally efficient. The application of ReLU ensures that the network can learn complex decision boundaries that are necessary for tasks like object detection.

In CNN-based object detection, the goal is to predict the class of an object and localize its position via a bounding box. The bounding box is parametrized by four coordinates: (x, y) for the center of the box, and w, h for the width and height. The task can be viewed as a twofold problem: (1) classify the object and (2) predict the bounding box that best encodes the object's spatial position. Mathematically, this requires the network to output both class probabilities and bounding box coordinates for each object within the image. The classification task is typically performed using a softmax function, which converts the network's raw output logits z_i for each class i into probabilities $P(y_i|r)$. The softmax function is defined as:

$$P(y_i|r) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \tag{16.26}$$

where k is the number of possible classes, z_i is the raw score for class i, and $P(y_i|r)$ is the probability that the region r belongs to class y_i. This function ensures that the predicted scores are valid probabilities that sum to one, which allows the network to make a probabilistic decision regarding the class of the object in each region. Simultaneously, the network must also predict the four parameters of the bounding box for each object. The network's predicted bounding box parameters are typically denoted as $\hat{B} = (\hat{x}, \hat{y}, \hat{w}, \hat{h})$, while the ground truth bounding box is denoted by $B = (x, y, w, h)$. The error between the predicted and true bounding boxes is quantified using a loss function, with the smooth L_1 loss being a commonly used metric for bounding box regression. The smooth L_1 loss for each parameter of the bounding box is defined as:

$$\mathcal{L}_{\text{bbox}} = \sum_{i=1}^{4} \text{SmoothL1}(B_i - \hat{B}_i) \tag{16.27}$$

The smooth L_1 function is defined as:

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{if } |x| \geq 1 \end{cases} \tag{16.28}$$

This loss function is used to reduce the impact of large errors, thereby making the training process more robust. The goal is to minimize this loss during the training phase to improve the network's ability to predict both the class and the bounding box of objects.

For training, a combined loss function is used that combines both the classification loss and the bounding box regression loss. The total loss function can be written as:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} \tag{16.29}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, typically computed using the cross-entropy between the predicted probabilities and the ground truth labels. The cross-entropy loss for classification is given by:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{k} y_i \log(\hat{y}_i) \tag{16.30}$$

where y_i is the true label, and $\hat{y}_i$ is the predicted probability for class i. The total objective function for training is therefore a weighted sum of the classification and bounding box regression losses, and the network is optimized to minimize this combined loss function. Object detection architectures like Region-based CNNs (R-CNNs) take a two-stage approach where the task is broken into generating region proposals and classifying these regions. Region Proposal Networks (RPNs) are employed to generate candidate regions $r_1, r_2, \ldots, r_n$, which are then passed through the network to obtain their feature representations. The bounding box refinement and classification for each proposal are then performed by a fully connected layer. The loss function for R-CNNs combines both classification and bounding box regression losses for each proposal, and the objective is to minimize:

$$\mathcal{L}_{\text{R-CNN}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} \tag{16.31}$$

Another popular architecture, YOLO (You Only Look Once), frames object detection as a single regression task. The image is divided into a grid of $S \times S$ cells, where each cell predicts the class probabilities and bounding box parameters. The output vector for each cell consists of:

$$\hat{y}_i = (x, y, w, h, c, P_1, P_2, \ldots, P_k) \tag{16.32}$$

where (x, y) are the coordinates of the bounding box center, w and h are the dimensions of the box, c is the confidence score, and $P_1, P_2, \ldots, P_k$ are the class probabilities. The total loss for YOLO combines the classification loss, bounding box regression loss, and confidence loss, which can be written as:

$$\mathcal{L}_{\text{YOLO}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} + \mathcal{L}_{\text{conf}} \tag{16.33}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, $\mathcal{L}_{\text{bbox}}$ is the bounding box regression loss, and $\mathcal{L}_{\text{conf}}$ is the confidence loss, which penalizes predictions with low confidence. This approach allows YOLO to make object detection predictions in a single pass through the network, enabling faster inference. The Single Shot Multibox Detector (SSD) improves on YOLO by generating bounding boxes at multiple feature scales, which allows for detecting objects of varying sizes. The loss function for SSD is similar to that of YOLO, comprising the classification loss and bounding box localization loss, given by:

$$\mathcal{L}_{\text{SSD}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{loc}} \tag{16.34}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, and $\mathcal{L}_{\text{loc}}$ is the smooth L_1 loss for bounding box regression. This multi-scale approach enhances the network's ability to detect objects at different levels of resolution, improving its robustness to objects of different sizes.

Thus, object detection in CNNs involves a sophisticated architecture of convolution, activation, pooling, and multi-stage loss functions that guide the network in accurately detecting and localizing objects in an image. The choice of architecture and loss function plays a critical role in the performance and efficiency of the detection system, with modern architectures like R-CNN, YOLO, and SSD each offering distinct advantages depending on the application requirements.

16.3 Real-World Applications

16.3.1 Medical Imaging

Literature Review: Yousif et. al. (2024) [205] applied CNNs for melanoma skin cancer detection, integrating a Binary Grey Wolf Optimization (GWO) algorithm to enhance feature selection. It demonstrates the effectiveness of deep learning in classifying dermatoscopic images and highlights feature extraction techniques for accurate classification. Rahman et. al. (2025) [206] gave a systematic review that covers CNN-based leukemia detection using medical imaging. The study compares different deep learning architectures such as ResNet, VGG, and EfficientNet, providing a benchmark for future studies. Joshi and Gowda (2025) [207] introduced an attention-guided Graph CNN (VSA-GCNN) for brain tumor segmentation and classification. It leverages spatial relationships within MRI scans to improve diagnostic accuracy. The use of graph neural networks (GNNs) combined with CNNs is a novel approach in medical imaging. Ng et al. (2025) [208] developed a CNN-based cardiac MRI analysis model to predict ischemic cardiomyopathy without contrast agents. It highlights the ability of deep learning models to extract diagnostic information from non-contrast images, reducing the need for invasive procedures. Nguyen et al. (2025) [209] presented a multi-view tumor region-adapted synthesis model for mammograms using CNNs. The approach enhances breast cancer detection by using 3D spatial feature extraction techniques, improving tumor localization and classification. Chen et. al. (2025) [210] explored CNN-based denoising for medical images using a penalized least squares (PLS) approach. The study applies deep learning for noise reduction in MRI scans, leading to improved clarity in low-signal-to-noise ratio (SNR) images. Pradhan et al. (2025) [211] discussed CNN-based diabetic retinopathy detection. It introduces an Atrous Residual U-Net architecture, enhancing image segmentation performance for early-stage diagnosis of retinal diseases. Örenç et al. (2025) [212] evaluated ensemble CNN models for adenoid hypertrophy detection in X-ray images. It demonstrates transfer learning and feature fusion techniques, which improve CNN-based medical diagnostics. Jiang et al. (2025) [213] introduced a cross-modal attention network for MRI image denoising, particularly effective when some imaging modalities are missing. It highlights cross-domain knowledge transfer using CNNs. Al-Haidri et. al. (2025) [214] developed a CNN-based framework for automatic myocardial fibrosis segmentation in cardiac MRI scans. It emphasizes quantitative feature extraction techniques that enhance precision in cardiac diagnostics.

Convolutional Neural Networks (CNNs) have become an indispensable tool in the field of medical imaging, driven by their ability to automatically learn spatial hierarchies of features directly from image data without the need for handcrafted feature extraction. The convolutional layers in CNNs are designed to exploit the spatial structure of the input data, making them particularly well-suited for tasks in medical imaging, where spatial relationships in images often encode critical diagnostic information. The fundamental building block of CNNs, the convolution operation, is mathematically expressed as

$$S(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n) \cdot K(m,n), \tag{16.35}$$

where $S(i,j)$ represents the value of the output feature map at position (i,j), $I(i,j)$ is the input image, $K(m,n)$ is the convolutional kernel (a learnable weight matrix), and k denotes the kernel radius (for example, $k = 1$ for a 3×3 kernel). This equation fundamentally captures how local patterns, such as edges, textures, and more complex features, are extracted by sliding the kernel across the image. The convolution operation is performed for each channel of a multi-channel input (e.g., RGB images or multi-modal medical images), and the results are summed across channels, leading to multi-dimensional feature maps. For a 3D input tensor, the convolution extends to include depth:

$$S(i,j,d') = \sum_{d=1}^{D} \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n, d) \cdot K(m,n,d), \tag{16.36}$$

where D is the depth of the input tensor, and d' is the depth index of the output feature map. CNNs incorporate nonlinear activation functions after convolutional layers to introduce nonlinearity into the model, allowing it to learn complex mappings. A commonly used activation function is the Rectified Linear Unit (ReLU), mathematically defined as

$$f(x) = \max(0, x). \tag{16.37}$$

This function ensures sparsity in the activations, which is advantageous for computational efficiency and generalization. More advanced activation functions, such as parametric ReLU (PReLU), extend this concept by allowing learnable parameters for the negative slope:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{if } x \leq 0, \end{cases} \tag{16.38}$$

where a is a learnable parameter. Pooling layers are employed in CNNs to downsample the spatial dimensions of feature maps, thereby reducing computational complexity and the risk of overfitting. Max pooling is defined mathematically as

$$P(i,j) = \max_{(m,n) \in \mathcal{R}} S(i+m, j+n), \tag{16.39}$$

where $\mathcal{R}$ is the pooling region (e.g., 2×2). Average pooling computes the mean value instead:

$$P(i,j) = \frac{1}{|\mathcal{R}|} \sum_{(m,n) \in \mathcal{R}} S(i+m, j+n). \tag{16.40}$$

In medical imaging, CNNs are widely used for image classification tasks such as detecting abnormalities (e.g., tumors, fractures, or lesions). Consider a classification problem where the input is a mammogram image, and the output is a binary label $y \in \{0, 1\}$, indicating benign or malignant. The CNN model outputs a probability score $\hat{y}$, computed as

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{16.41}$$

where z is the output of the final layer before the sigmoid activation. The binary cross-entropy loss function is then used to train the model:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]. \tag{16.42}$$

For image segmentation tasks, where the goal is to assign a label to each pixel, architectures such as U-Net are commonly used. U-Net employs an encoder-decoder structure, where the encoder extracts features through a series of convolutional and pooling layers, and the decoder reconstructs the image through upsampling and concatenation operations. The objective function for segmentation is often the Dice coefficient loss, defined as

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2\sum_i p_i g_i}{\sum_i p_i + \sum_i g_i}, \tag{16.43}$$

where p_i and g_i are the predicted and ground truth values for pixel i, respectively. In the context of image reconstruction, such as in magnetic resonance imaging (MRI), CNNs are used to reconstruct high-quality images from undersampled k-space data. The reconstruction problem is formulated as minimizing the difference between the reconstructed image I_{pred} and the ground truth I_{true}, often using the ℓ_2-norm:

$$\mathcal{L}_{\text{reconstruction}} = \|I_{\text{pred}} - I_{\text{true}}\|_2^2. \tag{16.44}$$

Generative adversarial networks (GANs) have also been applied to medical imaging, particularly for enhancing image resolution or synthesizing realistic images from noisy inputs. A GAN consists of a generator G and a discriminator D, where G learns to generate images $G(z)$ from latent noise z, and D distinguishes between real and fake images. The loss functions for G and D are given by

$$\mathcal{L}_D = -\mathbb{E}[\log D(x)] - \mathbb{E}[\log(1 - D(G(z)))], \tag{16.45}$$

$$\mathcal{L}_G = -\mathbb{E}[\log D(G(z))]. \tag{16.46}$$

Multi-modal imaging, where data from different modalities (e.g., MRI and PET) are combined, further highlights the utility of CNNs. For instance, feature maps from MRI and PET images are concatenated at intermediate layers to exploit complementary information, improving diagnostic accuracy. Attention mechanisms are often incorporated to focus on the most relevant regions of the image. For example, a spatial attention map A_s can be computed as

$$A_s = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot F + b_1) + b_2), \tag{16.47}$$

where F is the input feature map, W_1 and W_2 are learnable weight matrices, and b_1 and b_2 are biases. Despite their success, CNNs in medical imaging face challenges, including data scarcity and interpretability. Transfer learning addresses data scarcity by fine-tuning pre-trained models on small medical datasets. Techniques such as Grad-CAM provide interpretability by visualizing regions that influence the network's predictions. Mathematically, Grad-CAM computes the importance of a feature map A^k for a class c as

$$\alpha_k^c = \frac{1}{Z}\sum_{i,j}\frac{\partial y^c}{\partial A_{i,j}^k}, \tag{16.48}$$

where y^c is the score for class c and Z is a normalization constant. The class activation map is then obtained as

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right). \tag{16.49}$$

In summary, CNNs have transformed medical imaging by enabling automated and highly accurate analysis of complex medical images. Their applications span disease detection, segmentation, reconstruction, and multi-modal imaging, with continued advancements addressing challenges in data efficiency and interpretability. Their mathematical foundations and computational frameworks provide a robust basis for future innovations in this critical field.

16.3.2 Autonomous Vehicles

Literature Review: Ojala and Zhou (2024) [323] proposed a CNN-based approach for detecting and estimating object distances from thermal images in autonomous driving. They developed a deep convolutional model for distance estimation using a single thermal camera and introduced theoretical formulations for thermal imaging data preprocessing within CNN pipelines. Popordanoska and Blaschko (2025) [324] investigated the mathematical underpinnings of CNN calibration in high-risk domains, including autonomous vehicles. They analyzed the confidence calibration problem in CNNs used for self-driving perception and developed a Bayesian-inspired regularization approach to improve CNN decision reliability in autonomous driving. Alfieri et. al. (2024) [325] explored deep reinforcement learning (DRL) methods with CNNs for optimizing route planning in autonomous vehicles. They bridged CNN-based vision models with Deep Q-Learning, enabling adaptive path optimization in real-world driving conditions and established a novel theoretical connection between Q-learning and CNN-based object detection for autonomous navigation. Zanardelli (2025) [326] examined decision-making frameworks using CNNs in autonomous vehicle systems. He developed a statistical model integrating CNNs with reinforcement learning to improve self-driving car decision-making and provided a rigorous probabilistic analysis of how CNNs handle uncertainty in real-world driving environments. Norouzi et. al. (2025) [327] analyzed the role of transfer learning in CNN models for autonomous vehicle perception. They introduced pre-trained CNNs for vehicle object detection using multi-sensor data fusion and provided a rigorous theoretical justification for integrating Kalman filtering and Dempster-Shafer theory with CNNs. Wang et. al. (2024) [328] investigated the mathematics of uncertainty quantification in CNN-based perception models for self-driving cars. They used Bayesian CNNs to model uncertainty in semantic segmentation for autonomous driving and proposed a Dempster-Shafer theory-based fusion mechanism for combining multiple CNN outputs. Xia et. al. [329] integrated CNN-based perception models with reinforcement learning (RL) to improve autonomous vehicle trajectory tracking. They uses CNNs for lane detection and integrated them into a RL-based path planner. They also established a theoretical framework linking CNN-based scene recognition to control theory. Liu et. al. (2024) [330] introduced a CNN-based multi-view feature extraction framework for spatial-temporal analysis in self-driving cars. They developed a hybrid CNN-graph attention model to extract temporal driving patterns. They also made theoretical advancements in multi-view learning and feature fusion for CNNs in autonomous vehicle decision-making. Chakraborty and Deka (2025) [331] applied CNN-based multimodal sensor fusion to autonomous vehicles and UAVs for real-time navigation. They did theoretical analysis of CNN feature fusion mechanisms for real-time perception and developed mask region-based CNNs (Mask-RCNNs) for enhanced object recognition in autonomous navigation. Mirindi et. al. (2025) [332] investigated the role of CNNs and AI in smart autonomous transportation. They did theoretical discussion on the Unified Theory of AI Adoption in autonomous driving and introduced hybrid Recurrent Neural Networks (RNNs) and CNN architectures for vehicle trajectory prediction.

Convolutional Neural Networks (CNNs) are fundamental in the implementation of autonomous vehicles, forming the backbone of the perception and decision-making systems that allow these vehicles to interpret and respond to their environment. At the core of a CNN is the convolution operation, which mathematically transforms an input image or signal into a feature map, allowing the extraction of spatial hierarchies of information. The convolution operation in its continuous form is defined as:

$$s(t) = \int_{-\infty}^{\infty} x(\tau) w(t-\tau)\, d\tau, \tag{16.50}$$

where $x(\tau)$ represents the input, $w(t-\tau)$ is the filter or kernel, and $s(t)$ is the output feature. In the discrete domain, especially for image processing, this operation can be written as:

$$S(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} X(i+m, j+n) \cdot W(m,n), \tag{16.51}$$

where $X(i,j)$ denotes the pixel intensity at coordinate (i,j) of the input image, and $W(m,n)$ represents the convolutional kernel values. This operation enables the detection of local patterns such as edges, corners, or textures, which are then aggregated across layers to recognize complex features like shapes and objects. In the context of autonomous vehicles, CNNs process sensor data from cameras, LiDAR, and radar to identify critical features such as other vehicles, pedestrians, road signs, and lane boundaries. For object detection, CNN-based architectures such as YOLO (You Only Look Once) and Faster R-CNN employ a backbone network like ResNet, which uses successive convolutional layers to extract hierarchical features from the input image. The object detection task involves two primary outputs: bounding box coordinates and object class probabilities. Mathematically, bounding box regression is modeled as a multi-task learning problem. The loss function for bounding box regression is often formulated as:

$$L_{\text{reg}} = \sum_{i=1}^{N} \sum_{j \in \{x,y,w,h\}} \text{SmoothL1}(t_{ij} - \hat{t}_{ij}), \tag{16.52}$$

where t_{ij} and $\hat{t}_{ij}$ are the ground-truth and predicted bounding box parameters (e.g., center coordinates x, y and dimensions w, h). Simultaneously, the classification loss, typically cross-entropy, is computed as:

$$L_{\text{cls}} = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{16.53}$$

where $y_{i,c}$ is a binary indicator for whether the object at index i belongs to class c, and $p_{i,c}$ is the predicted probability. The total loss function is a weighted combination:

$$L_{\text{total}} = \alpha L_{\text{reg}} + \beta L_{\text{cls}}. \tag{16.54}$$

Semantic segmentation, another critical task, requires pixel-level classification to assign a label (e.g., road, vehicle, pedestrian) to each pixel in an image. Fully Convolutional Networks (FCNs) or U-Net architectures are commonly used for this purpose. These architectures utilize an encoder-decoder structure where the encoder extracts spatial features, and the decoder reconstructs the spatial resolution to generate pixel-wise predictions. The loss function for semantic segmentation is a sum over all pixels and classes, given as:

$$L = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{16.55}$$

where $y_{i,c}$ is the ground-truth binary label for pixel i and class c, and $p_{i,c}$ is the predicted probability. Advanced architectures also employ skip connections to preserve high-resolution spatial information, enabling sharper segmentation boundaries.

Depth estimation is essential for autonomous vehicles to understand the 3D structure of their surroundings. CNNs are used to predict depth maps from monocular images or stereo pairs. The depth estimation process is modeled as a regression problem, where the loss function is designed to minimize the difference between the predicted depth $\hat{d}_i$ and the ground-truth depth d_i. A commonly used loss function for this task is the scale-invariant loss:

$$L_{\text{scale-inv}} = \frac{1}{n} \sum_{i=1}^{n} \left(\log d_i - \log \hat{d}_i \right)^2 - \frac{1}{n^2} \left(\sum_{i=1}^{n} \left(\log d_i - \log \hat{d}_i \right) \right)^2. \tag{16.56}$$

This loss ensures that the relative depth differences are minimized, which is critical for accurate 3D reconstruction. Lane detection, another critical application, uses CNNs to detect road lanes and boundaries. The task often involves predicting the lane markings as polynomial curves. CNNs process the input image to extract lane features, and post-processing involves fitting a curve, such as:

$$y = ax^2 + bx + c, \tag{16.57}$$

where a, b, c are the coefficients predicted by the network. The fitting process minimizes an error function, typically the sum of squared differences between the detected lane points and the curve:

$$E = \sum_{i=1}^{N} (y_i - (ax_i^2 + bx_i + c))^2. \tag{16.58}$$

In autonomous vehicles, these CNN tasks are integrated into an end-to-end pipeline. The input data from cameras, LiDAR, and radar is first processed using CNNs to extract features relevant to the vehicle's perception. The outputs, including object detections, semantic maps, depth maps, and lane boundaries, are then passed to the planning module, which computes the vehicle's trajectory. For instance, detected objects provide information about obstacles, while lane boundaries guide path planning algorithms. The planning process involves solving optimization problems where the objective function incorporates constraints from the CNN outputs. For example, a trajectory optimization problem may minimize a cost function:

$$J = \int_0^T \left(w_1 \dot{x}^2 + w_2 \dot{y}^2 + w_3 c(t)\right) \, dt, \tag{16.59}$$

where $\dot{x}$ and $\dot{y}$ are the lateral and longitudinal velocities, and $c(t)$ is a collision penalty based on object detections.

In conclusion, CNNs provide the computational framework for perception tasks in autonomous vehicles, enabling real-time interpretation of complex sensory data. By leveraging mathematical principles of convolution, loss optimization, and hierarchical feature extraction, CNNs transform raw sensor data into actionable insights, paving the way for safe and efficient autonomous navigation.

16.4 Popular CNN Architectures

Literature Review: Choudhury et. al. (2024) [333] presented a comparative theoretical study of CNN architectures, including AlexNet, VGG, and ResNet, for satellite-based aircraft identification. They analyzed the architectural differences and learning strategies used in VGG, AlexNet, and ResNet and theoretically explained how VGG's depth, AlexNet's feature extraction, and ResNet's residual learning contribute to CNN advancements. Almubarok and Rosiani (2024) [334] discussed the computational efficiency of CNN architectures, particularly focusing on AlexNet, VGG, and ResNet in comparison to MobileNetV2. They established theoretical efficiency trade-offs between depth, parameter count, and accuracy in AlexNet, VGG, and ResNet and highlighted ResNet's advantage in optimization due to skip connections, compared to AlexNet and VGG's traditional deep structures. Ding (2024) [335] explored CNN architectures (AlexNet, VGG, and ResNet) for medical image classification, particularly in Traditional Chinese Medicine (TCM). He introduced ResNet-101 with Squeeze-and-Excitation (SE) blocks, expanding theoretical understanding of deep feature representations in CNNs and discussed VGG's weight-sharing strategy and AlexNet's layered feature extraction, improving classification accuracy. He et. al. (2015) [336] introduced Residual Learning, demonstrating how deep CNNs benefit from identity mappings to tackle vanishing gradients. They formulated the mathematical justification of residual blocks in deep networks and Established the theoretical backbone of ResNet's identity mapping for deep optimization. Simonyan and Zisserman (2014) [148] presented the VGG architecture, which demonstrates how depth improvement enhances feature extraction. They developed the theoretical formulation of increasing CNN depth and its impact on feature hierarchies and provided an analytical framework for receptive field expansion in deep CNNs. Krizhevsky et. al. (2012) [337] introduced AlexNet, the first CNN model to achieve state-of-the-art performance in ImageNet classification. They introduced ReLU activation as a breakthrough in CNN training and established dropout regularization theory, preventing overfitting in deep networks. Sultana et. al. (2019) [338] compared the feature extraction strategies of AlexNet, VGG, and ResNet for object recognition. They gave theoretical explanation of hierarchical feature learning in CNN architectures and examined VGG's use of small convolutional filters and how it impacts feature map depth. Sattler et. al. (2019) [339] investigated the fundamental limitations of CNN architectures such as AlexNet, VGG, and ResNet. They established formal constraints on convolutional filters in CNNs and developed a theoretical model for CNN generalization error in classification tasks.

16.4.1 AlexNet

The **AlexNet Convolutional Neural Network (CNN)** is a deep learning model that operates on raw pixel values to perform image classification. Given an input image, represented as a 3D tensor $\mathbf{I}_0 \in \mathbb{R}^{H \times W \times C}$, where H is the height, W is the width, and C represents the number of input channels (typically $C = 3$ for RGB images), the network performs a series of operations, such as convolutions, activation functions, pooling, and fully connected layers, to transform this input into a final output vector $\mathbf{y} \in \mathbb{R}^K$, where K is the number of output classes. The objective of AlexNet is to minimize a loss function that measures the discrepancy between the predicted output and the true label, typically using the **cross-entropy loss** function.

At the heart of AlexNet's architecture are the **convolutional layers**, which are designed to learn local patterns in the image by convolving a set of filters over the input image. Specifically, the first convolutional layer performs a convolution of the input image $\mathbf{I}_0$ with a set of filters $\mathbf{W}_1^{(k)} \in \mathbb{R}^{F_1 \times F_1 \times C}$, where F_1 is the size of the filter and C is the number of channels in the input. The convolution operation for a given filter $\mathbf{W}_1^{(k)}$ and input image $\mathbf{I}_0$ at position (i, j) is defined as:

$$Y_1^{(k)}(i,j) = \sum_{u=1}^{F_1} \sum_{v=1}^{F_1} \sum_{c=1}^{C} W_1^{(k)}(u,v,c) \cdot I_0(i+u-1, j+v-1, c) + b_1^{(k)} \tag{16.60}$$

where $b_1^{(k)}$ is the bias term for the k-th filter, and the output of this convolution is a feature map $Y_1^{(k)}(i,j)$ that captures the response of the filter at each spatial location (i, j). The result of this convolution operation is a set of feature maps $Y_1^{(k)} \in \mathbb{R}^{H' \times W'}$, where the dimensions of the output are $H' = H - F_1 + 1$ and $W' = W - F_1 + 1$ if no padding is applied. Subsequent to the convolutional operation, the output feature maps $Y_1^{(k)}$ are passed through a **ReLU (Rectified Linear Unit)** activation function, which introduces non-linearity into the network. The ReLU function is defined as:

$$\text{ReLU}(z) = \max(0, z) \tag{16.61}$$

This function transforms negative values in the feature map $Y_1^{(k)}$ into zero, while leaving positive values unchanged, thus allowing the network to model complex, non-linear patterns in the data. The output of the ReLU activation function is denoted by $A_1^{(k)}(i,j) = \text{ReLU}(Y_1^{(k)}(i,j))$. Following the activation function, a **max-pooling** operation is performed to downsample the feature maps and reduce their spatial dimensions. Given a pooling window of size $P \times P$, the max-pooling operation computes the maximum value in each window, which is mathematically expressed as:

$$Y_1^{\text{pool}}(i,j) = \max\left(A_1^{(k)}(i',j') : (i',j') \in \text{pooling window}\right) \tag{16.62}$$

where $A_1^{(k)}$ is the feature map after ReLU, and the resulting pooled output $Y_1^{\text{pool}}(i,j)$ has reduced spatial dimensions, typically $H'' = \frac{H'}{P}$ and $W'' = \frac{W'}{P}$. This operation helps retain the most important features while discarding irrelevant spatial details, which makes the network more robust to small translations in the input image. The convolutional and pooling operations are repeated across multiple layers, with each layer learning progressively more complex patterns from the input data. In the second convolutional layer, for example, we convolve the feature maps from the first layer $A_1^{(k)}$ with a new set of filters $\mathbf{W}_2^{(k)} \in \mathbb{R}^{F_2 \times F_2 \times K_1}$, where K_1 is the number of feature maps produced by the first convolutional layer. The convolution for the second layer is expressed as:

$$Y_2^{(k)}(i,j) = \sum_{u=1}^{F_2} \sum_{v=1}^{F_2} \sum_{c=1}^{K_1} W_2^{(k)}(u,v,c) \cdot A_1^{(c)}(i+u-1, j+v-1) + b_2^{(k)} \tag{16.63}$$

This process is iterated for each subsequent convolutional layer, where each new set of filters learns higher-level features, such as edges, textures, and object parts. The activation maps produced by each convolutional layer are passed through the ReLU activation function, and max-pooling is applied after each convolutional layer to reduce the spatial dimensions.

After the last convolutional layer, the feature maps are **flattened** into a 1D vector $\mathbf{a}_f \in \mathbb{R}^N$, where N is the total number of activations across all channels and spatial dimensions. This flattened vector is then passed to **fully connected (FC) layers** for classification. Each fully connected layer performs a linear transformation, followed by a non-linear activation. The output of the i-th neuron in the fully connected layer is given by:

$$z_i = \sum_{j=1}^{N} W_{ij} \cdot a_f(j) + b_i \tag{16.64}$$

where W_{ij} is the weight connecting neuron j in the previous layer to neuron i in the current layer, and b_i is the bias term. The output of the fully connected layer is a vector of class scores $\mathbf{z} \in \mathbb{R}^K$, which represents the unnormalized log-probabilities of the input image belonging to each class. To convert these scores into a valid probability distribution, the **softmax** function is applied:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{16.65}$$

The softmax function ensures that the output values are in the range $[0,1]$ and sum to 1, thus representing a probability distribution over the K classes. The final output of the network is a probability vector $\hat{\mathbf{y}} \in \mathbb{R}^K$, where each element $\hat{y}_i$ corresponds to the predicted probability that the input image belongs to class i. To train the AlexNet model, the network minimizes the **cross-entropy loss** function between the predicted probabilities $\hat{\mathbf{y}}$ and the true labels $\mathbf{y}$, which is given by:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{16.66}$$

where y_i is the true label (1 if the image belongs to class i, 0 otherwise), and $\hat{y}_i$ is the predicted probability for class i. The goal of training is to adjust the weights W and biases b in the network to minimize this loss. The parameters of the network are updated using **gradient descent**. To compute the gradients, the **backpropagation** algorithm is used. The gradient of the loss with respect to the weights W in a fully connected layer is given by:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W} \tag{16.67}$$

where $\frac{\partial L}{\partial z}$ is the gradient of the loss with respect to the output of the layer, and $\frac{\partial z}{\partial W}$ is the gradient of the output with respect to the weights. These gradients are then used to update the weights using the gradient descent update rule:

$$W \leftarrow W - \eta \cdot \frac{\partial L}{\partial W} \tag{16.68}$$

where η is the learning rate. This process is repeated iteratively for each layer of the network.

Regularization techniques such as **dropout** are often applied to prevent overfitting during training. Dropout involves randomly setting a fraction of the activations to zero during each training step, which helps prevent the network from relying too heavily on any one feature and encourages the model to learn more robust features. Once trained, the AlexNet model can be used to classify new images by passing them through the network and selecting the class with the highest probability. The combination of convolutional layers, ReLU activations, pooling, fully connected layers, and softmax activation makes AlexNet a powerful and efficient architecture for image classification tasks.

16.4.2 ResNet

At the heart of the ResNet architecture lies the notion of *residual learning*, where instead of learning the direct transformation $\mathbf{y} = f(\mathbf{x}; \mathbf{W})$, the network learns the residual function $\mathcal{F}(\mathbf{x}, \mathbf{W})$, i.e., the difference between the input and output. The network output $\mathbf{y}$ can therefore be expressed as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \mathbf{W}) + \mathbf{x} \tag{16.69}$$

This formulation represents the core difference from traditional neural networks where the model learns a mapping directly from the input $\mathbf{x}$ to the output $\mathbf{y}$. The introduction of the identity shortcut connection $\mathbf{x}$ introduces a powerful mechanism by which the network can learn the residual, and if the optimal residual transformation is the identity function, the network can essentially learn $\mathbf{y} = \mathbf{x}$, improving optimization. This reduces the challenge of training deeper networks, where deep layers often lead to vanishing gradients, because the gradient can propagate directly through these shortcuts, bypassing intermediate layers.

Let's formalize this residual learning. Let the input to the residual block be $\mathbf{x}_l$ and the output $\mathbf{y}_l$. In a conventional neural network, the transformation from input to output at the l-th layer would be:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) \tag{16.70}$$

where $\mathcal{F}$ represents the function learned by the layer, parameterized by $\mathbf{W}_l$. In contrast, for ResNet, the output is the sum of the learned residual function $\mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)$ and the input $\mathbf{x}_l$ itself, yielding:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{x}_l \tag{16.71}$$

This addition of the identity shortcut connection enables the network to bypass layers if needed, facilitating the learning process and addressing the vanishing gradient issue. To formalize the optimization problem, we define the residual learning objective as the minimization of the loss function $\mathcal{L}$ with respect to the parameters $\mathbf{W}_l$:

$$\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{y}_i, \mathbf{t}_i) \tag{16.72}$$

where N is the number of training samples, $\mathbf{t}_i$ are the target outputs, and $\mathcal{L}_i$ is the loss for the i-th sample. The training process involves adjusting the parameters $\mathbf{W}_l$ via gradient descent, which in turn requires the gradients of the loss function with respect to the network parameters. The gradient of $\mathcal{L}$ with respect to $\mathbf{W}_l$ can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}_i}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{W}_l} \tag{16.73}$$

Since the residual block adds the input directly to the output, the derivative of the output with respect to the weights $\mathbf{W}_l$ is given by:

$$\frac{\partial \mathbf{y}_l}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)}{\partial \mathbf{W}_l} \tag{16.74}$$

Now, let's explore how this addition of the residual connection directly influences the backpropagation process. In a traditional feedforward network, the backpropagated gradients for each layer depend solely on the output of the preceding layer. However, in a residual network, the gradient flow is enhanced because the identity mapping $\mathbf{x}_l$ is directly passed to the subsequent layer. This ensures that the gradients will not be lost as the network deepens, a phenomenon that becomes critical in very deep networks. The gradient with respect to the loss $\mathcal{L}$ at layer l is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_l} \cdot \frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_l} \tag{16.75}$$

Since $\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{x}_l$, the derivative of $\mathbf{y}_l$ with respect to $\mathbf{x}_l$ is:

$$\frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_l} = \mathbf{I} + \frac{\partial \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)}{\partial \mathbf{x}_l} \tag{16.76}$$

where $\mathbf{I}$ is the identity matrix. This ensures that the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_l}$ can propagate more easily through the network, as it is now augmented by the identity matrix term. Thus, this term helps preserve the gradient's magnitude during backpropagation, solving the vanishing gradient problem that typically arises in deep networks. Furthermore, to ensure that the dimensions of the input and output of a residual block match, especially when the number of channels changes, ResNet introduces *projection shortcuts*. These are used when the dimensionality of $\mathbf{x}_l$ and $\mathbf{y}_l$ do not align, typically through a 1×1 convolution. The projection shortcut modifies the residual block's output to be:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{W}_x \cdot \mathbf{x}_l \tag{16.77}$$

where $\mathbf{W}_x$ is a convolutional filter, and $\mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)$ is the residual transformation. The introduction of the 1×1 convolution ensures that the input $\mathbf{x}_l$ is mapped to the appropriate dimensionality, while still benefiting from the residual learning framework. The ResNet architecture can be extended by stacking multiple residual blocks. For a network with L layers, the output after passing through the entire network can be written recursively as:

$$\mathbf{y}^{(L)} = \mathbf{x} + \mathcal{F}(\mathbf{y}^{(L-1)}; \mathbf{W}_L) \tag{16.78}$$

where $\mathbf{y}^{(L-1)}$ is the output after $L - 1$ layers. The recursive nature of this formula ensures that the network's output is built layer by layer, with each layer contributing a transformation relative to the input passed to it. Mathematically, the gradient of the loss function with respect to the parameters in deep residual networks can be expressed recursively, where each layer's gradient involves contributions from the identity shortcut connection. This facilitates the training of very deep networks by maintaining a stable and consistent flow of gradients during the backpropagation process.

Thus, the *Residual Neural Network (ResNet)* significantly improves the trainability of deep neural networks by introducing residual learning, allowing the network to focus on learning the difference between the input and output rather than the entire transformation. This approach, combined with identity shortcut connections and projection shortcuts for dimensionality matching, ensures that gradients flow effectively through the network, even in very deep architectures. The resulting ResNet architecture has been proven to enable the training of networks with hundreds of layers, yielding impressive performance on a wide range of tasks, from image classification to semantic segmentation, while mitigating issues such as vanishing gradients. Through its recursive structure and rigorous mathematical formulation, ResNet has become a foundational architecture in modern deep learning.

16.4.3 VGG

The **Visual Geometry Group (VGG) Convolutional Neural Network (CNN)**, introduced by Simonyan and Zisserman in 2014, presents a detailed exploration of the effect of depth on the performance of deep neural networks, specifically within the context of computer vision tasks such as image classification. The VGG architecture is grounded in the hypothesis that deeper networks, when constructed with small, consistent convolutional kernels, are more capable of capturing hierarchical patterns in data, particularly in the domain of visual recognition. In contrast to other CNN architectures, VGG prioritizes the usage of small 3×3 convolution filters (with a stride of 1) stacked in increasing depth, rather than relying on larger filters (e.g., 5×5 or 7×7), thus offering computational benefits without sacrificing representational power. This design choice inherently encourages sparse local receptive fields, which ensures a richer learning capacity when extended across deeper layers.

Let $I \in \mathbb{R}^{H \times W \times C}$ represent an input image of height H, width W, and C channels, where the channels correspond to different color representations (e.g., RGB for $C = 3$). For the convolution operation applied at a particular layer k, the output feature map $O^{(k)}$ can be computed by convolving the input I with a set of kernels $K^{(k)}$ corresponding to the k-th layer. The convolution for each spatial location i, j can be described as:

$$O_{i,j}^{(k)} = \sum_{u=1}^{k_h} \sum_{v=1}^{k_w} \sum_{c'=1}^{C_{\text{in}}} K_{u,v,c',c}^{(k)} I_{i+u,j+v,c'} + b_c^{(k)} \tag{16.79}$$

where $O_{i,j}^{(k)}$ is the output value at location (i, j) of the feature map for the k-th filter, $K_{u,v,c',c}^{(k)}$ is the u, v-th spatial element of the c'-to-c filter in layer k, and $b_c^{(k)}$ represents the bias term for the output channel c. The convolutional layer's kernel $K^{(k)}$ is typically initialized with small values and learned during training, while the bias $b^{(k)}$ is added to shift the activation of the neuron. A key aspect of the VGG architecture is that these convolution layers are consistently followed by non-linear **ReLU (Rectified Linear Unit)** activation functions, which introduce local non-linearity to the model. The ReLU function is mathematically defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{16.80}$$

This transformation is applied element-wise, ensuring that negative values are mapped to zero, which, as an effect, activates only positive feature responses. The non-linearity introduced by ReLU aids the network in learning complex patterns and overcoming issues such as vanishing gradients that often arise in deeper networks. In VGG, the network is constructed by stacking these convolutional layers with ReLU activations. Each convolution layer is followed by **max-pooling** operations, typically with 2×2 filters and a stride of 2. Max-pooling reduces the spatial dimensions of the feature maps and extracts the most significant features from each region of the image. The max-pooling operation is mathematically expressed as:

$$O_{i,j} = \max_{(u,v) \in P} I_{i+u,j+v} \tag{16.81}$$

where P is the pooling window, and $O_{i,j}$ is the pooled value at position (i, j). The pooling operation performs downsampling, ensuring translation invariance while retaining the most prominent features. The effect of this pooling operation is to reduce computational complexity, lower the number of parameters, and make the network invariant to small translations and distortions in the input image. The architecture of VGG typically culminates in a series of **fully connected (FC) layers** after several convolutional and pooling layers have extracted relevant features from the input image. Let the output of the final convolutional layer, after flattening, be denoted as $X \in \mathbb{R}^d$, where d represents the dimensionality of the feature vector obtained by flattening the last convolutional feature map. The fully connected layers then transform this vector into the output, as expressed by:

$$\mathbf{O} = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{16.82}$$

where $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is the weight matrix of the fully connected layer, $\mathbf{b} \in \mathbb{R}^{d'}$ is the bias vector, and $\mathbf{O} \in \mathbb{R}^{d'}$ is the output vector. The output vector $\mathbf{O}$ represents the unnormalized scores for each of the d' possible classes in a classification task. This is typically followed by the application of a **softmax** function to convert these raw scores into a probability distribution:

$$\sigma(o_i) = \frac{e^{o_i}}{\sum_{j=1}^{d'} e^{o_j}} \tag{16.83}$$

where o_i is the score for class i, and the softmax function ensures that the outputs are positive and sum to one, facilitating their interpretation as class probabilities. This softmax function is a crucial step in multi-class classification tasks as it normalizes the output into a probabilistic format. During the training phase, the model minimizes the **cross-entropy loss** between the predicted probabilities and the actual class labels, often represented as one-hot encoded vectors. The cross-entropy loss is given by:

$$L = -\sum_{i=1}^{d'} y_i \log(p_i) \tag{16.84}$$

where y_i is the true label for class i in one-hot encoded form, and p_i is the predicted probability for class i. This loss function is the appropriate objective for classification tasks, as it measures the difference between the true and predicted probability distributions. The optimization of the parameters in the VGG network is carried out using **stochastic gradient descent (SGD)** or its variants. The weight update rule in gradient descent is:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L \tag{16.85}$$

where η is the learning rate, and $\nabla_{\mathbf{W}} L$ is the gradient of the loss with respect to the weights. The gradient is computed through **backpropagation**, applying the chain rule of derivatives to propagate errors backward through the network, updating the weights at each layer based on the contribution of each parameter to the final output error.

A key advantage of the VGG architecture lies in its use of smaller, deeper layers compared to previous networks like AlexNet, which used larger convolution filters. By using multiple small kernels (such as 3×3), the VGG network can create richer representations without exponentially increasing the number of parameters. The depth of the network, achieved by stacking these small convolution filters, enables the model to extract increasingly abstract and hierarchical features from the raw pixel data. Despite its success, VGG's computational demands are relatively high due to the large number of parameters, especially in the fully connected layers. The fully connected layers, which connect every neuron in one layer to every neuron in the next, account for a significant portion of the model's total parameters. To mitigate this limitation, later architectures, such as **ResNet**, introduced **skip connections**, which allow gradients to flow more efficiently through the network, thus enabling even deeper architectures without incurring the same computational costs. Nevertheless, the VGG network set an important precedent in the design of deep convolutional networks, demonstrating the power of deep architectures and the effectiveness of small convolutional filters. The model's simplicity and straightforward design have influenced subsequent architectures, reinforcing the notion that deeper models, when carefully constructed, can achieve exceptional performance on complex tasks like image classification, despite the challenges posed by computational cost and model complexity.

17 Recurrent Neural Networks (RNNs)

Literature Review: Schmidhuber (2015) [114] provided an extensive historical perspective on neural networks, including RNNs. Schmidhuber describes key architectures such as Long Short-Term Memory (LSTM) and their importance in solving the vanishing gradient problem. He also explains fundamental learning algorithms for training RNNs and provides insights into applications like sequence prediction and speech recognition. Lipton et. al. (2015) [264] offers a rigorous critique of RNNs and their various implementations. The authors discuss the fundamental challenges of training RNNs, including long-range dependencies and computational inefficiencies. The paper also presents benchmarks comparing different architectures like vanilla RNNs, LSTMs, and GRUs. offers a rigorous critique of RNNs and their various implementations. The authors discuss the fundamental challenges of training RNNs, including long-range dependencies and computational inefficiencies. The paper also presents benchmarks comparing different architectures like vanilla RNNs, LSTMs, and GRUs. Pascanu et. al. (2013) [265] formally analyzes why training RNNs is difficult, particularly focusing on the vanishing and exploding gradient problem. The authors propose gradient clipping as a practical solution and discuss ways to improve training efficiency for RNNs. Goodfellow et. al. (2016) [112] in their book book dedicates an entire chapter to recurrent neural networks, discussing their theoretical foundations, backpropagation through time (BPTT), and key architectures such as LSTMs and GRUs. It also provides mathematical derivations of optimization techniques used in training deep RNNs. Jaeger (2001) [266] introduced the Echo State Network (ESN), an alternative recurrent architecture that requires only the output weights to be trained. The ESN approach has become highly influential in RNN research, particularly for solving stability and efficiency problems. Hochreiter and Schmidhuber (1997) [267] introduced the LSTM architecture, which solves the vanishing gradient problem in RNNs by incorporating memory cells with gating mechanisms. LSTMs are now a standard in sequence modeling tasks, such as speech recognition and natural language processing. Kawakami (2008) [268] provided a deep dive into supervised learning techniques for RNNs, particularly for sequence labeling problems. Graves discusses Connectionist Temporal Classification (CTC), a popular loss function for RNN-based speech and handwriting recognition. Bengio et. al. (1994) [269] mathematically proved why RNNs struggle with learning long-term dependencies. It identifies the root causes of the vanishing and exploding gradient problems, setting the stage for future architectures like LSTMs. Bhattamishra et. al. (2020) [270] rigorously compared the theoretical capabilities of RNNs and Transformers. The authors analyze expressiveness, memory retention, and training efficiency, providing insights into why Transformers are increasingly replacing RNNs in NLP. Siegelmann (1993) [271] provided a rigorous theoretical treatment of RNNs, analyzing their convergence properties, stability conditions, and computational complexity. It discusses mathematical frameworks for understanding RNN generalization and optimization challenges.

17.1 Key Concepts

Recurrent Neural Networks (RNNs) are a class of neural architectures specifically designed for processing sequential data, leveraging their recursive structure to model temporal dependencies. At the core of an RNN lies the concept of a hidden state $\mathbf{h}_t \in \mathbb{R}^m$, which evolves over time as a function of the current input $\mathbf{x}_t \in \mathbb{R}^n$ and the previous hidden state $\mathbf{h}_{t-1}$. This evolution is governed by the recurrence relation:

$$\mathbf{h}_t = f_h \left(\mathbf{W}_{xh} \mathbf{x}_t + \mathbf{W}_{hh} \mathbf{h}_{t-1} + \mathbf{b}_h \right), \tag{17.1}$$

where $\mathbf{W}_{xh} \in \mathbb{R}^{m \times n}$ is the input-to-hidden weight matrix, $\mathbf{W}_{hh} \in \mathbb{R}^{m \times m}$ is the hidden-to-hidden weight matrix, $\mathbf{b}_h \in \mathbb{R}^m$ is the bias vector, and f_h is a non-linear activation function, typically

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{17.2}$$

or the rectified linear unit $\text{ReLU}(x) = \max(0, x)$. The recursive nature of this update equation ensures that $\mathbf{h}_t$ inherently encodes information about the sequence $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$, allowing the network to maintain a dynamic representation of past inputs. The output $\mathbf{y}_t \in \mathbb{R}^o$ at time t is computed as:

$$\mathbf{y}_t = f_y \left(\mathbf{W}_{hy} \mathbf{h}_t + \mathbf{b}_y \right), \tag{17.3}$$

where $\mathbf{W}_{hy} \in \mathbb{R}^{o \times m}$ is the hidden-to-output weight matrix, $\mathbf{b}_y \in \mathbb{R}^o$ is the output bias, and f_y is an activation function such as the softmax function:

$$f_y(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{o} e^{z_j}} \tag{17.4}$$

for classification tasks. Expanding the recurrence relation iteratively, the hidden state at time t can be expressed as:

$$\mathbf{h}_t = f_h \left(\mathbf{W}_{xh} \mathbf{x}_t + \mathbf{W}_{hh} f_h \left(\mathbf{W}_{xh} \mathbf{x}_{t-1} + \mathbf{W}_{hh} f_h \left(\ldots f_h \left(\mathbf{W}_{xh} \mathbf{x}_1 + \mathbf{W}_{hh} \mathbf{h}_0 + \mathbf{b}_h \right) + \mathbf{b}_h \right) + \mathbf{b}_h \right) + \mathbf{b}_h \right). \tag{17.5}$$

This expansion illustrates the depth of temporal dependency captured by the network and highlights the computational challenges of maintaining long-term memory. Specifically, the gradient of the loss function L, given by:

$$L = \sum_{t=1}^{T} \ell \left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}} \right), \tag{17.6}$$

with $\ell \left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}} \right)$ representing a task-specific loss such as cross-entropy:

$$\ell \left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}} \right) = - \sum_{i=1}^{o} \mathbf{y}_t^{\text{true}}(i) \log \mathbf{y}_t(i), \tag{17.7}$$

is computed through backpropagation through time (BPTT). The gradient of L with respect to $\mathbf{W}_{hh}$, for instance, is given by:

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \ell_t}{\partial \mathbf{h}_t} \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_{hh}}, \tag{17.8}$$

where $\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$ represents the chain of derivatives from time step k to t. Unlike feedforward neural networks, where each input is processed independently, RNNs maintain a hidden state h_t that acts as a dynamic memory, evolving recursively as the input sequence progresses. Formally, given an input sequence $\{x_1, x_2, \ldots, x_T\}$, where $x_t \in \mathbb{R}^n$ represents the input vector at time t, the hidden state $h_t \in \mathbb{R}^m$ is updated via the recurrence relation:

$$h_t = f_h(W_{xh} x_t + W_{hh} h_{t-1} + b_h), \tag{17.9}$$

where $W_{xh} \in \mathbb{R}^{m \times n}$, $W_{hh} \in \mathbb{R}^{m \times m}$, and $b_h \in \mathbb{R}^m$ are learnable parameters, and f_h is a nonlinear activation function such as tanh or ReLU. The recursive structure inherently allows the hidden state h_t to encode the entire history of the sequence up to time t. The output $y_t \in \mathbb{R}^o$ at each time step is computed as:

$$y_t = f_y(W_{hy}h_t + b_y), \tag{17.10}$$

where $W_{hy} \in \mathbb{R}^{o\times m}$ and $b_y \in \mathbb{R}^o$ are additional learnable parameters, and f_y is an optional output activation function, such as the softmax function for classification. To elucidate the recursive dynamics, we can expand h_t explicitly in terms of the initial hidden state h_0 and all previous inputs $\{x_1, \ldots, x_t\}$:

$$h_t = f_h(W_{xh}x_t + W_{hh}f_h(W_{xh}x_{t-1} + W_{hh}f_h(\ldots f_h(W_{xh}x_1 + W_{hh}h_0 + b_h) + b_h) + b_h) + b_h). \tag{17.11}$$

This nested structure highlights the temporal dependencies and the potential challenges in training, such as the vanishing and exploding gradient problems. During training, the loss function L, which aggregates the discrepancies between the predicted outputs y_t and the ground truth y_t^{true}, is typically defined as:

$$L = \sum_{t=1}^{T} \ell(y_t, y_t^{\text{true}}), \tag{17.12}$$

where ℓ is a task-specific loss function, such as the mean squared error (MSE)

$$\ell(y, y^{\text{true}}) = \frac{1}{2}\|y - y^{\text{true}}\|^2 \tag{17.13}$$

for regression or the cross-entropy loss for classification. To optimize L, gradient-based methods are employed, requiring the computation of derivatives of L with respect to all parameters, such as W_{xh}, W_{hh}, and b_h. Using backpropagation through time (BPTT), the gradient of L with respect to W_{hh} is expressed as:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \ell_t}{\partial h_t} \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \frac{\partial h_k}{\partial W_{hh}}. \tag{17.14}$$

Here,

$$\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{17.15}$$

is the product of Jacobian matrices that encode the influence of h_k on h_t. The Jacobian $\frac{\partial h_j}{\partial h_{j-1}}$ itself is given by:

$$\frac{\partial h_j}{\partial h_{j-1}} = W_{hh} \odot f_h'(a_j), \tag{17.16}$$

where

$$a_j = W_{xh}x_j + W_{hh}h_{j-1} + b_h, \tag{17.17}$$

and $f_h'(a_j)$ denotes the elementwise derivative of the activation function. The repeated multiplication of these Jacobians can lead to exponential growth or decay of the gradients, depending on the spectral radius $\rho(W_{hh})$. Specifically, if $\rho(W_{hh}) > 1$, gradients explode, whereas if $\rho(W_{hh}) < 1$, gradients vanish, severely hampering the training process for long sequences. To address these issues, modifications such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) introduce gating mechanisms that explicitly regulate the flow of information. In LSTMs, the cell state c_t, governed by additive dynamics, prevents vanishing gradients. The cell state is updated as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{17.18}$$

where f_t is the forget gate, i_t is the input gate, and U_c, W_c, and b_c are learnable parameters.

17.2 Sequence Modeling and Long Short-Term Memory (LSTM) and GRUs

Literature Review: Potter and Egon (2024) [387] provided an extensive study of RNNs and their enhancements (LSTM and GRU) for time-series forecasting. The authors conduct an empirical comparison between these architectures and analyze their effectiveness in capturing long-term dependencies in sequential data. The study concludes that GRUs are computationally efficient but slightly less expressive than LSTMs, whereas standard RNNs suffer from vanishing gradients. Yatkin et. al. (2025) [388] introduced a topological perspective to RNNs, including LSTM and GRU, to address inconsistencies in real-world applications. The authors propose stability-enhancing mechanisms to improve RNN performance in finance and climate modeling. Their results show that topologically-optimized GRUs outperform traditional LSTMs in maintaining memory over long sequences. Saifullah (2024) [389] applied LSTM and GRU networks to biomedical image classification (chicken egg fertility detection). The paper demonstrates that GRU's simpler architecture leads to faster convergence while LSTMs achieve slightly higher accuracy due to better memory retention. The results highlight domain-specific strengths of LSTM vs. GRU, particularly in handling sparse feature representations. Alonso (2024) [390] rigorously explored the mathematical foundations of RNNs, LSTMs, and GRUs. The author provides a deep analysis of gating mechanisms, vanishing gradient solutions, and optimization techniques that improve sequence modeling. A theoretical comparison is drawn between hidden state dynamics in GRUs vs. LSTMs, supporting their application in NLP and time-series forecasting. Tu et. al. (2024) [391] in a medical AI study evaluates LSTMs and GRUs for predicting patient physiological metrics during sedation. The authors find that LSTMs retain more long-term dependencies in time-series medical data, making them suitable for patient monitoring, while GRUs are preferable for real-time predictions due to their lower computational overhead. Zuo et. al. (2025) [392] applied hybrid GRUs for predicting customer movements in stores using real-time location tracking. The authors propose a modified GRU-LSTM hybrid model that achieves state-of-the-art accuracy in trajectory prediction. The study demonstrates that GRUs alone may lack fine-grained memory retention, but a hybrid approach improves forecasting ability. Lima et. al. (2025) [393] developed an industrial AI application that demonstrated the efficiency of GRUs in process optimization. The study finds that GRUs outperform LSTMs in real-time predictive control of steel slab heating, showcasing their efficiency in applications where faster computations are required. Khan et. al. (2025) [394] integrated LSTMs with statistical ARIMA models to improve wind power forecasting. They demonstrate that hybrid LSTM-ARIMA models outperform standalone RNNs in handling weather-related sequential data, which is highly volatile. Guo and Feng (2024) [395] in an environmental AI study proposed a temporal attention-enhanced LSTM model to predict greenhouse climate variables. The research introduces a novel position-aware LSTM architecture that improves multi-step forecasting, which is critical for precision agriculture. Abdelhamid (2024) [396] explored IoT-based energy forecasting using deep RNN architectures, including LSTM and GRU. The study concludes that GRUs provide faster inference speeds but LSTMs capture more accurate long-range dependencies, making them more reliable for complex forecasting.

Sequence modeling in Recurrent Neural Networks (RNNs) represents a powerful framework for capturing temporal dependencies in sequential data, enabling the learning of both short-term and long-term patterns. The primary characteristic of RNNs lies in their recurrent architecture, where the hidden state h_t at time step t is updated as a function of both the current input x_t and the hidden state at the previous time step h_{t-1}. Mathematically, this recurrent relationship can be expressed as:

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h) \tag{17.19}$$

Here, W_h and W_x are weight matrices corresponding to the previous hidden state h_{t-1} and the current input x_t, respectively, while b_h is a bias term. The function $f(\cdot)$ is a non-linear activation function, typically chosen as the hyperbolic tangent tanh or rectified linear unit (ReLU). The output y_t at each time step is derived from the hidden state h_t through a linear transformation, followed by a non-linear activation, yielding:

$$y_t = g(W_y h_t + b_y) \tag{17.20}$$

where W_y is the weight matrix connecting the hidden state to the output space, and b_y is the associated bias term. The function $g(\cdot)$ is generally a softmax activation for classification tasks or a linear activation for regression problems. The key feature of this structure is the interdependence of the hidden state across time steps, allowing the model to capture the history of past inputs and produce predictions that incorporate temporal context. Training an RNN involves minimizing a loss function L, which quantifies the discrepancy between the predicted outputs y_t and the true outputs y_t^{true} across all time steps. A common loss function used in classification tasks is the cross-entropy loss, while regression tasks often utilize mean squared error. To optimize the parameters of the network, the model employs **Backpropagation Through Time (BPTT)**, a variant of the standard backpropagation algorithm adapted for sequential data. The primary challenge in BPTT arises from the recurrent nature of the network, where the hidden state at each time step depends on the previous hidden state. The gradient of the loss function with respect to the hidden state at time step t is computed recursively, reflecting the temporal structure of the model. The chain rule is applied to compute the gradient of the loss with respect to the hidden state:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} + \sum_{t'=t+1}^{T} \frac{\partial L}{\partial h_{t'}} \cdot \frac{\partial h_{t'}}{\partial h_t} \tag{17.21}$$

Here, $\frac{\partial L}{\partial y_t}$ is the gradient of the loss with respect to the output, and $\frac{\partial y_t}{\partial h_t}$ represents the Jacobian of the output with respect to the hidden state. The second term in this expression corresponds to the accumulated gradients propagated from future time steps, incorporating the temporal dependencies across the entire sequence. This recursive gradient calculation allows for updating the weights and biases of the RNN, adjusting them to minimize the total error across the sequence. The gradients of the loss function with respect to the parameters of the network, such as W_h, W_x, and W_y, are computed using the chain rule. For example, the gradient of the loss with respect to W_x is:

$$\frac{\partial L}{\partial W_x} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \cdot x_t^\top \tag{17.22}$$

This captures the contribution of each input to the overall error at all time steps, ensuring that the model learns the correct relationships between inputs and hidden states. Similarly, the gradients with respect to W_h and b_h account for the recurrence in the hidden state, enabling the model to adjust its internal parameters in response to the sequential nature of the data. Despite their theoretical elegance, RNNs face significant practical challenges during training, primarily due to the **vanishing gradients problem**. This issue arises when the gradients propagate through many time steps, causing them to decay exponentially, especially when using activation functions like tanh. As a result, the influence of distant time steps diminishes, making it difficult for the network to learn long-term dependencies. The mathematical manifestation of this problem is seen in the norm of the Jacobian matrices associated with the hidden state updates. If the spectral radius of the weight matrices W_h is close to or greater than 1, the gradients can either vanish or explode, leading to unstable training dynamics. To mitigate this issue, several solutions have been proposed, including the use of Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which introduce gating mechanisms to better control the flow of information through the network. LSTMs, for example, incorporate a memory cell C_t, which allows the network to store information over long periods of time. The update rules for the LSTM are governed by three gates:

the forget gate f_t, the input gate i_t, and the output gate o_t, which control how much of the previous memory and new information to retain. The equations governing the LSTM are:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{17.23}$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{17.24}$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{17.25}$$

$$\tilde{C}_t = \tanh(W_C h_{t-1} + U_C x_t + b_C) \tag{17.26}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{17.27}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{17.28}$$

In these equations, the forget gate f_t determines how much of the previous memory cell C_{t-1} to retain, the input gate i_t governs how much new information to store in the candidate memory cell $\tilde{C}_t$, and the output gate o_t controls how much of the memory cell should influence the current output. The LSTM's architecture allows for the maintenance of long-term dependencies by selectively forgetting or retaining information, effectively alleviating the vanishing gradient problem and enabling the network to learn from longer sequences. The GRU, an alternative to the LSTM, simplifies this architecture by combining the forget and input gates into a single update gate z_t, and introduces a reset gate r_t to control the influence of the previous hidden state. The GRU's update rules are:

$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z) \tag{17.29}$$

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r) \tag{17.30}$$

$$\tilde{h}_t = \tanh(W h_{t-1} + U x_t + b) \tag{17.31}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{17.32}$$

Here, z_t controls the amount of the previous hidden state h_{t-1} to retain, and r_t determines how much of the previous hidden state should influence the candidate hidden state $\tilde{h}_t$. The GRU's simplified structure still allows it to effectively capture long-range dependencies while being computationally more efficient than the LSTM.

In summary, sequence modeling in RNNs involves a series of recurrent updates to the hidden state, driven by both the current input and the previous hidden state, and is trained via backpropagation through time. The introduction of specialized gating mechanisms in LSTMs and GRUs alleviates issues such as vanishing gradients, enabling the networks to learn and maintain long-term dependencies. Through these advanced architectures, RNNs can effectively model complex temporal relationships, making them powerful tools for tasks such as time-series prediction, natural language processing, and sequence generation.

17.3 Applications in Natural Language Processing

Literature Review: Yang et. al. (2020) [377] explored the effectiveness of deep learning models, including RNNs, for sentiment analysis in e-commerce platforms. It emphasizes how RNN architectures, including LSTMs and GRUs, outperform traditional NLP techniques by capturing sequential dependencies in customer reviews. The study provides empirical evidence demonstrating the superior accuracy of RNNs in analyzing consumer sentiment. Manikandan et. al. (2025) [378] investigated how RNNs can improve spam detection in email filtering. By leveraging recurrent structures, the study demonstrates how RNNs effectively identify patterns in email text that indicate spam or phishing attempts. It also compares RNN-based models with other ML approaches, highlighting the robustness of RNNs in handling contextual word sequences. Isiaka et. al. (2025) [379] examined AI technologies, particularly deep learning models, for predictive healthcare applications. It highlights how RNNs can analyze patient records and medical reports using NLP techniques. The study shows that RNN-based NLP models enhance medical diagnostics and decision-making by extracting meaningful insights from unstructured text data. Petrov et. al. (2025) [380] discussed the role of RNNs in emotion classification from textual data, an essential NLP task. The paper evaluates various RNN-based architectures, including BiLSTMs, to enhance the accuracy of emotion recognition in social media texts and chatbot responses. Liang (2025) [381] focused on the application of RNNs in educational settings, specifically for automated grading and feedback generation. The study presents an RNN-based NLP system capable of analyzing student responses, providing real-time assessments, and generating contextual feedback. Jin (2025) [382] explored how RNNs optimize text generation tasks related to pharmaceutical education. It demonstrates how NLP-powered RNN models generate high-quality textual summaries from medical literature, ensuring accurate knowledge dissemination in the pharmaceutical industry. McNicholas et. al. (2025) [383] investigated how RNNs facilitate clinical decision-making in critical care by extracting insights from unstructured medical text. The research highlights how RNN-based NLP models enhance patient care by predicting outcomes based on clinical notes and physician reports. Abbas and Khammas (2024) [384] introduced an RNN-based NLP technique for detecting malware in IoT networks. The study illustrates how RNN classifiers process logs and textual patterns to identify malicious software, making RNNs crucial in cybersecurity applications. Kalonia and Upadhyay (2025) [385] applied RNNs to software fault prediction using NLP techniques. It shows how recurrent networks analyze bug reports and software documentation to predict potential failures in software applications, aiding developers in proactive debugging. Han et. al. (2025) [386] discussed RNN applications in conversational AI, focusing on chatbots and virtual assistants. The study presents an RNN-driven NLP model for improving dialogue management and user interactions, significantly enhancing the responsiveness of AI-powered chat systems.

Recurrent Neural Networks (RNNs) are deep learning architectures that are explicitly designed to handle sequential data, a key feature that makes them indispensable for applications in Natural Language Processing (NLP). The mathematical foundation of RNNs lies in their ability to process sequences of inputs, $x_1, x_2, \ldots, x_T$, where T denotes the length of the sequence. At each time step t, the network updates its hidden state, h_t, using both the current input x_t and the previous hidden state h_{t-1}. This recursive relationship is represented mathematically by the following equation:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{17.33}$$

Here, σ is a nonlinear activation function such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU), W_h is the weight matrix associated with the previous hidden state h_{t-1}, W_x is the weight matrix associated with the current input x_t, and b is a bias term. The nonlinearity introduced by σ allows the network to learn complex relationships between the input and the output. The output y_t at each time step is computed as:

$$y_t = W_y h_t + c \tag{17.34}$$

where W_y is the weight matrix corresponding to the output and c is the bias term for the output. The output y_t is then used to compute the predicted probability distribution over possible outputs at each time step, typically through a softmax function for classification tasks:

$$P(y_t|h_t) = \text{softmax}(W_y h_t + c) \tag{17.35}$$

In NLP tasks such as language modeling, the objective is to predict the next word in a sequence given all previous words. The RNN is trained to estimate the conditional probability distribution $P(w_t|w_1, w_2, \ldots, w_{t-1})$ of the next word w_t based on the previous words. The full likelihood of the sequence $w_1, w_2, \ldots, w_T$ can be written as:

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t|w_1, w_2, \ldots, w_{t-1}) \tag{17.36}$$

For an RNN, this conditional probability is modeled by recursively updating the hidden state and generating a probability distribution for each word. At each time step, the probability of the next word is computed as:

$$P(w_t|h_{t-1}) = \text{softmax}(W_y h_t + c) \tag{17.37}$$

The network is trained by minimizing the negative log-likelihood of the true word sequence:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(w_t|h_{t-1}) \tag{17.38}$$

This loss function guides the optimization of the weight matrices W_h, W_x, and W_y to maximize the likelihood of the correct word sequences. As the network learns from large datasets, it develops the ability to predict words based on the context provided by previous words in the sequence. A key extension of RNNs in NLP is machine translation, where one sequence of words in one language is mapped to another sequence in a target language. This is typically modeled using sequence-to-sequence (Seq2Seq) architectures, which consist of two RNNs: the encoder and the decoder. The encoder RNN processes the input sequence $x_1, x_2, \ldots, x_T$, updating its hidden state at each time step:

$$h_t^{\text{enc}} = \sigma(W_h^{\text{enc}} h_{t-1}^{\text{enc}} + W_x^{\text{enc}} x_t + b^{\text{enc}}) \tag{17.39}$$

The final hidden state h_T^{enc} of the encoder is passed to the decoder as its initial hidden state. The decoder RNN generates the target sequence $y_1, y_2, \ldots, y_T$ by updating its hidden state at each time step, using both the previous hidden state h_{t-1}^{dec} and the previous output y_{t-1}:

$$h_t^{\text{dec}} = \sigma(W_h^{\text{dec}} h_{t-1}^{\text{dec}} + W_x^{\text{dec}} y_{t-1} + b^{\text{dec}}) \tag{17.40}$$

The decoder produces a probability distribution over the target vocabulary at each time step:

$$P(y_t|h_t^{\text{dec}}) = \text{softmax}(W_y^{\text{dec}} h_t^{\text{dec}} + c^{\text{dec}}) \tag{17.41}$$

The training of the Seq2Seq model is based on minimizing the cross-entropy loss function:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t|h_t^{\text{dec}}) \tag{17.42}$$

This ensures that the network learns to map input sequences to output sequences. By training on a large corpus of paired sentences, the Seq2Seq model learns to translate sentences from one language to another, with the encoder capturing the context of the input sentence and the decoder generating the translated sentence.

RNNs are also effective in sentiment analysis, a task where the goal is to classify the sentiment of a sentence (positive, negative, or neutral). Given a sequence of words $x_1, x_2, \ldots, x_T$, the RNN processes each word sequentially, updating its hidden state:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{17.43}$$

After processing the entire sentence, the final hidden state h_T is used to classify the sentiment. The output is obtained by applying a softmax function to the final hidden state:

$$y = \text{softmax}(W_y h_T + c) \tag{17.44}$$

where W_y is the weight matrix associated with the output layer. The network is trained to minimize the cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y|h_T) \tag{17.45}$$

This allows the RNN to classify the overall sentiment of the sentence by learning the relationships between words and sentiment labels. Sentiment analysis is useful for applications such as customer feedback analysis, social media monitoring, and opinion mining. In Named Entity Recognition (NER), RNNs are used to identify and classify named entities, such as people, locations, and organizations, in a text. The RNN processes each word x_t in the sequence, updating its hidden state at each time step:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{17.46}$$

The output at each time step is a probability distribution over possible entity labels:

$$P(y_t|h_t) = \text{softmax}(W_y h_t + c) \tag{17.47}$$

The network is trained to minimize the cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t|h_t) \tag{17.48}$$

By learning to classify each word with the appropriate entity label, the RNN can perform information extraction tasks, such as identifying people, organizations, and locations in text. This is crucial for applications such as document categorization, knowledge graph construction, and question answering. In speech recognition, RNNs are used to transcribe spoken language into text. The input to the RNN consists of a sequence of acoustic features, such as Mel-frequency cepstral coefficients (MFCCs), which are extracted from the audio signal. At each time step t, the RNN updates its hidden state:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{17.49}$$

The output at each time step is a probability distribution over phonemes or words:

$$P(w_t|h_t) = \text{softmax}(W_y h_t + c) \tag{17.50}$$

The network is trained by minimizing the negative log-likelihood:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(w_t|h_t) \tag{17.51}$$

By learning the mapping between acoustic features and corresponding words or phonemes, the RNN can transcribe speech into text, which is fundamental for applications such as voice assistants, transcription services, and speech-to-text systems.

In summary, RNNs are powerful tools for processing sequential data in NLP tasks such as machine translation, sentiment analysis, named entity recognition, and speech recognition. Their ability to process input sequences in a time-dependent manner allows them to capture long-range dependencies, making them well-suited for complex tasks in NLP and beyond. However, challenges such as the vanishing and exploding gradient problems necessitate the use of more advanced architectures, like LSTMs and GRUs, to enhance their performance in real-world applications.

18 Advanced Architectures

18.1 Transformers and Attention Mechanisms

Literature Review: Vaswani et. al. [340] introduced the Transformer architecture, replacing recurrent models with a fully attention-based framework for sequence processing. They formulated the self-attention mechanism, mathematically defining query-key-value (QKV) transformations. They proved scalability advantages over RNNs, showing O(1) parallelization benefits and introduced multi-head attention, enabling contextualized embeddings. Nannepagu et. al. [341] explored hybrid AI architectures integrating Transformers with deep reinforcement learning (DQN). They developed a theoretical framework for transformer-augmented reinforcement learning and discussed how self-attention refines feature representations for financial time-series prediction. Rose et. al. [342] investigated Vision Transformers (ViTs) for cybersecurity applications, examining attention-based anomaly detection. They theoretically compared self-attention with CNN feature extraction and proposed a new loss function for attention weight refinement in cybersecurity detection models. Buehler [343] explored the theoretical interplay between Graph Neural Networks (GNNs) and Transformer architectures. They developed isomorphic self-attention, which preserves graph topological information and introduced graph-structured positional embeddings within Transformer attention. Tabibpour and Madanizadeh [344] investigated Set Transformers as a theoretical extension of Transformers for high-dimensional dynamic systems and introduced permutation-invariant self-attention mechanisms to replace standard Transformers in decision-making tasks and theoretically formalized attention mechanisms for non-sequential data. Kim et. al. (2024) [310] developed a Transformer-based anomaly detection framework for video surveillance. They formalized a new spatio-temporal self-attention mechanism to detect anomalies in videos and extended standard Transformer architectures to handle high-dimensional video data. Li and Dong [345] examined Transformer-based attention mechanisms for wireless communication networks. They introduced hybrid spatial and temporal attention layers for large-scale MIMO channel estimation and provided a rigorous mathematical proof of attention-based signal recovery. Asefa and Assabie [346] investigated language-specific adaptations of Transformer-based translation models. They introduced attention mechanism regularization for low-resource language translation and analyzed the impact of different positional encoding strategies on translation quality. Liao and Chen [347] applied transformer architectures to deepfake detection, analyzing self-attention mechanisms for facial feature analysis. They theoretically compared CNNs and ViTs for forgery detection and introduced attention-head dropout to improve robustness against adversarial attacks. Jiang et. al. [348] proposed a novel Transformer-based approach for medical imaging reconstruction. They introduced Spatial and Channel-wise Transformer (SCFormer) for enhanced attention-based feature aggregation and theoretically extended contrastive learning to Transformer encoders.

The Transformer model is an advanced neural network architecture fundamentally defined by the self-attention mechanism, which enables global context-aware computations on sequential data. The model processes an input sequence represented by

$$\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}, \tag{18.1}$$

where n denotes the sequence length and d_{model} the embedding dimensionality. Each token in this sequence is projected into three learned spaces—queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$—using the trainable matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$, such that

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{18.2}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$, with d_k being the dimensionality of queries and keys. The pairwise similarity between tokens is determined by the dot product $\mathbf{Q}\mathbf{K}^\top$, scaled by the factor $\frac{1}{\sqrt{d_k}}$ to ensure numerical stability, yielding the raw attention scores:

$$\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}, \tag{18.3}$$

where $\mathbf{S} \in \mathbb{R}^{n \times n}$. These scores are normalized using the softmax function, producing the attention weights $\mathbf{A}$, where

$$A_{ij} = \frac{\exp(S_{ij})}{\sum_{k=1}^{n} \exp(S_{ik})}, \tag{18.4}$$

ensuring $\sum_{j=1}^{n} A_{ij} = 1$. The output of the attention mechanism is computed as a weighted sum of the values:

$$\mathbf{Z} = \mathbf{A}\mathbf{V}, \tag{18.5}$$

where $\mathbf{Z} \in \mathbb{R}^{n \times d_v}$, with d_v being the dimensionality of the value vectors. This process can be expressed compactly as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{18.6}$$

Multi-head attention extends this mechanism by splitting $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into h distinct heads, each operating in its subspace. For the i-th head:

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) \tag{18.7}$$

where $\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q$, $\mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K$, $\mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V$. The outputs of all heads are concatenated and linearly transformed:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O, \tag{18.8}$$

where $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. This architecture enables the model to capture multiple types of relationships simultaneously. Positional encodings are added to the input embeddings $\mathbf{X}$ to preserve sequence order. These encodings $\mathbf{P} \in \mathbb{R}^{n \times d_{\text{model}}}$ are defined as:

$$P_{(pos,2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad P_{(pos,2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \tag{18.9}$$

ensuring unique representations for each position pos and dimension index i. The feedforward network (FFN) applies two dense layers with an intermediate ReLU activation:

$$\text{FFN}(\mathbf{z}) = \max(0, \mathbf{z}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{18.10}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, and $d_{\text{ff}} > d_{\text{model}}$. Residual connections and layer normalization are applied throughout to stabilize training, with the output given by

$$\mathbf{H}_{\text{out}} = \text{LayerNorm}(\mathbf{H}_{\text{in}} + \text{FFN}(\mathbf{H}_{\text{in}})). \tag{18.11}$$

Training optimizes the cross-entropy loss over the output distribution:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, \mathbf{x}), \tag{18.12}$$

where $P(y_t \mid y_{<t}, \mathbf{x})$ is modeled using the softmax over the logits $\mathbf{z}_t\mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}$, with parameters $\mathbf{W}_{\text{out}}, \mathbf{b}_{\text{out}}$. The Transformer achieves a complexity of $O(n^2 d_k)$ per attention layer due to the computation of $\mathbf{Q}\mathbf{K}^\top$, yet its parallelization capabilities render it more efficient than recurrent networks. This mathematical formalism, coupled with innovations like sparse attention and dynamic programming, has solidified the Transformer as the cornerstone of modern sequence modeling tasks. While this quadratic complexity poses challenges for very long sequences, it allows for greater parallelization compared to RNNs, which require $O(n)$ sequential steps. Furthermore, the memory complexity of $O(n^2)$ for storing attention weights can be mitigated using sparse approximations or hierarchical attention structures. The Transformer architecture's flexibility and effectiveness stem from its ability to handle diverse tasks by appropriately modifying its components. For example, in Vision Transformers (ViTs), the input sequence is formed by flattening image patches, and the positional encodings capture spatial relationships. In contrast, in sequence-to-sequence tasks like translation, the cross-attention mechanism enables the decoder to focus on relevant parts of the encoder's output.

In conclusion, the Transformer represents a paradigm shift in neural network design, replacing recurrence with attention and enabling unprecedented scalability and performance. The rigorous mathematical foundation of attention mechanisms, combined with the architectural innovations of multi-head attention, positional encoding, and feedforward layers, underpins its success across domains.

18.2 Generative Adversarial Networks (GANs)

Literature Review: Goodfellow et. al. [349] in their landmark paper introduced Generative Adversarial Networks (GANs), where a generator and a discriminator compete in a minimax game. They established the theoretical foundation of adversarial learning and developed the mathematical formulation of GANs using game theory. They also introduced non-cooperative minimax optimization in deep learning. Chappidi and Sundaram [350] extended GANs with graph neural networks (GNNs) for complex real-world perception tasks. They theoretically integrated GANs with reinforcement learning for self-improving models and developed dual Q-learning mechanisms that enhance GAN convergence stability. Joni [351] provided a comprehensive theoretical overview of GAN-based generative models for advanced image synthesis. They formalized GAN loss functions and their optimization challenges and introduced progressive growing GANs as a solution for high-resolution image generation. Li et. al. (2024) [305] extended GANs to materials science, optimizing the crystal structure prediction process. They developed a GAN framework for molecular modeling and demonstrates GANs in scientific simulations beyond computer vision tasks. Sekhavat (2024) [299] analyzed the philosophy and theoretical basis of GANs in artistic image generation. He discussed GANs from a cognitive science perspective and established a link between GAN training and computational aesthetics. Kalaiarasi and Sudharani (2024) [352] examined GAN-based image steganography, optimizing data hiding techniques using adversarial training. They extended the theoretical properties of adversarial training to security applications and demonstrated how GANs can minimize perceptual distortion in data hiding. Arjmandi-Tash and Mansourian (2024) [353] explored GANs in scientific computing, generating realistic photodetector datasets. They demonstrated GANs as a theoretical tool for synthetic data augmentation and formulated a probabilistic approach to GAN loss functions for sensor modeling. Gao (2024) [354] bridged the gap between GANs and Partial Differential Equations (PDEs) in physics-informed learning. He established a theoretical framework for solving PDEs using GAN-based architectures and developed a new loss function combining adversarial and variational methods. Hisama et. al. [355] applied GANs to computational chemistry, generating new alloy catalyst structures. They introduced Wasserstein GANs (WGANs) for molecular design and used GAN-generated latent spaces to predict catalyst activity. Wang and Zhang (2024) [356] proposed an improved GAN framework for medical image segmentation. They developed a novel attention-enhanced GAN for robust segmentation and provided a mathematical formulation for adversarial segmentation loss functions.

Generative Adversarial Networks (GANs) are an intricate mathematical framework designed to model complex probability distributions by leveraging a competitive dynamic between two neural networks, the generator G and the discriminator D. These networks are parametrized by weights $\theta_G \in \Theta_G$ and $\theta_D \in \Theta_D$, and their interaction is mathematically formulated as a two-player zero-sum game. The generator $G : \mathbb{R}^d \to \mathbb{R}^n$ maps latent variables $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$, where $p_{\mathbf{z}}$ is a prior probability distribution (commonly uniform or Gaussian), to a synthetic data sample $\hat{\mathbf{x}} = G(\mathbf{z})$. The discriminator $D : \mathbb{R}^n \to [0, 1]$ assigns a probability score $D(\mathbf{x})$ indicating whether $\mathbf{x}$ originates from the true data distribution $p_{\text{data}}(\mathbf{x})$ or the generated distribution $p_g(\mathbf{x})$, implicitly defined as the pushforward measure of $p_{\mathbf{z}}$ under G, i.e., $p_g = G_{\#}p_{\mathbf{z}}$. The optimization problem governing GANs is expressed as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{18.13}$$

where $\mathbb{E}$ denotes the expectation operator. This objective seeks to maximize the discriminator's ability to distinguish between real and generated samples while simultaneously minimizing the generator's ability to produce samples distinguishable from real data. For a fixed generator G, the optimal discriminator D^* is obtained by maximizing $V(D, G)$, yielding

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{18.14}$$

Substituting D^* back into the value function simplifies it to

$$V(D^*, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim p_g}\left[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\right]. \tag{18.15}$$

This expression is equivalent to minimizing the Jensen-Shannon (JS) divergence between p_{data} and p_g, defined as

$$\text{JS}(p_{\text{data}}\|p_g) = \frac{1}{2}\text{KL}(p_{\text{data}}\|M) + \frac{1}{2}\text{KL}(p_g\|M), \tag{18.16}$$

where $M = \frac{1}{2}(p_{\text{data}} + p_g)$ and $\text{KL}(p\|q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}$ is the Kullback-Leibler divergence. At the Nash equilibrium, $p_g = p_{\text{data}}$, the JS divergence vanishes, and $D^*(\mathbf{x}) = \frac{1}{2}$ for all $\mathbf{x}$. The gradient updates during training are derived using stochastic gradient descent. For the discriminator, the gradients are given by

$$\nabla_{\theta_D} V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\nabla_{\theta_D} \log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\nabla_{\theta_D} \log(1 - D(G(\mathbf{z})))]. \tag{18.17}$$

Training Generative Adversarial Networks (GANs) involves iterative updates to the parameters θ_D of the discriminator and θ_G of the generator. The discriminator's parameters are updated via gradient ascent to maximize the value function $V(D, G)$, while the generator's parameters are updated via gradient descent to minimize the same value function. Denoting the gradients of D and G with respect to their parameters as ∇_{θ_D} and ∇_{θ_G}, the updates are given by:

$$\theta_D \leftarrow \theta_D + \eta_D \nabla_{\theta_D}\left[\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\right], \tag{18.18}$$

and

$$\theta_G \leftarrow \theta_G - \eta_G \nabla_{\theta_G}\left[\mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\right]. \tag{18.19}$$

In practice, to address issues of vanishing gradients, an alternative loss function for the generator is often used, defined as:

$$-\mathbb{E}_{z\sim p_z}[\log D(G(z))]. \tag{18.20}$$

This modification ensures stronger gradient signals when the discriminator is performing well, effectively improving the generator's training dynamics. For the generator, the gradients in the original formulation are expressed as

$$\nabla_{\theta_G} V(D, G) = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \left[\nabla_{\theta_G} \log(1 - D(G(\mathbf{z})))\right], \tag{18.21}$$

but due to vanishing gradients when $D(G(\mathbf{z}))$ is near 0, the non-saturating generator loss is preferred:

$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log D(G(\mathbf{z}))]. \tag{18.22}$$

The convergence of GANs is inherently linked to the properties of $D^*(\mathbf{x})$ and the alignment of p_g with p_{data}. However, mode collapse and training instability are frequently observed due to the non-convex nature of the objective functions. Wasserstein GANs (WGANs) address these issues by replacing the JS divergence with the Wasserstein-1 distance, defined as

$$W(p_{\text{data}}, p_g) = \inf_{\gamma\in\Pi(p_{\text{data}}, p_g)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}[\|\mathbf{x} - \mathbf{y}\|], \tag{18.23}$$

where $\Pi(p_{\text{data}}, p_g)$ is the set of all couplings of p_{data} and p_g. Using Kantorovich-Rubinstein duality, the Wasserstein distance is reformulated as

$$W(p_{\text{data}}, p_g) = \sup_{\|f\|_L\leq 1} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_g}[f(\mathbf{x})], \tag{18.24}$$

where f is a 1-Lipschitz function. To enforce the Lipschitz constraint, gradient penalties are applied, ensuring that $\|\nabla f(\mathbf{x})\| \leq 1$.

The mathematical framework of GANs integrates elements from game theory, optimization, and information geometry. Their training involves solving a high-dimensional non-convex game, where theoretical guarantees for convergence are challenging due to saddle points and complex interactions between G and D. Nevertheless, GANs represent a mathematically elegant paradigm for generative modeling, with ongoing research extending their theoretical and practical capabilities.

18.3 Autoencoders and Variational Autoencoders

Literature Review: Zhang et. al. (2024) [303] explored a theoretical connection between VAEs and rate-distortion theory in image compression. They established a mathematical framework linking probabilistic autoencoding to lossy image compression and introduced hierarchical variational inference for improving generative modeling capacity. Wang and Huang (2025) [304] developed a formal mathematical proof of convergence in over-parameterized VAEs. They established rigorous mathematical limits for training VAEs and introduced Neural Tangent Kernel (NTK) theory to study how VAEs behave under over-parameterization. Li et. al. (2024) [305] extended VAEs to materials science, optimizing crystal structure prediction. They developed a VAE-based molecular modeling framework and demonstrates the role of generative models beyond image-based applications. Huang (2024) [306] reviewed key techniques in VAEs, GANs, and Diffusion Models for image generation. They analyzed probabilistic modeling in VAEs compared to diffusion-based methods and also established a theoretical hierarchy of generative models. Chenebuah (2024) [307] investigated Autoencoders for energy materials simulation and molecular property prediction. They introduced a novel AE-VAE hybrid model for physical simulations and established a theoretical link between Density Functional Theory (DFT) and Autoencoders. Furth et. al. (2024) [308] explored Graph Neural Networks (GNNs) and VAEs for predicting chemical properties. They established theoretical properties of VAEs for graph-based learning and extended Autoencoders to chemical reaction prediction. Gong et. al. [309] investigated Conditional Variational Autoencoders

(CVAEs) for material design. They introduced new loss functions for conditional generative modeling and theoretically proved how VAEs can optimize material selection. Kim et. al. [310] uses Transformer-based Autoencoders (AEs) for video anomaly detection. They established theoretical improvements of AEs for time-series anomaly detection and used spatio-temporal Autoencoder embeddings to capture anomalies in videos. Albert et. al. (2024) [311] compared Kernel Learning Embeddings (KLE) and Variational Autoencoders for dimensionality reduction. They introduced VAE-based models for atmospheric modeling and established a mathematical comparison between VAEs and kernel-based models. Sharma et. al. (2024) [312] explored practical applications of Autoencoders in network intrusion detection. They established Autoencoders as robust feature extractors for anomaly detection and provided a formal study of Autoencoder latent space representations.

An **Autoencoder (AE)** is an unsupervised learning model that attempts to learn a compact representation of the input data $\mathbf{x} \in \mathbb{R}^d$ in a lower-dimensional latent space. This model consists of two primary components: an encoder function f_{θ_e} and a decoder function f_{θ_d}. The encoder $f_{\theta_e} : \mathbb{R}^d \to \mathbb{R}^l$ maps the input data $\mathbf{x}$ to a latent code $\mathbf{z}$, where $l \ll d$, representing the compressed information. The decoder $f_{\theta_d} : \mathbb{R}^l \to \mathbb{R}^d$ then reconstructs the input from this latent code, producing an approximation $\hat{\mathbf{x}}$. The loss function typically used to train the autoencoder is the reconstruction loss, often formulated as the Mean Squared Error (MSE):

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2. \tag{18.25}$$

The optimization procedure seeks to minimize the reconstruction error over the dataset D, assuming a distribution $p(\mathbf{x})$ over the input data $\mathbf{x}$. The objective is to learn the optimal parameters θ_e and θ_d, by solving the following optimization problem:

$$\min_{\theta_e, \theta_d} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \left[\|\mathbf{x} - f_{\theta_d}(f_{\theta_e}(\mathbf{x}))\|_2^2 \right]. \tag{18.26}$$

This formulation drives the encoder-decoder architecture towards learning a latent representation that preserves key features of the input data, allowing it to be efficiently reconstructed. The solution to this problem is typically pursued via **stochastic gradient descent (SGD)**, where gradients of the loss with respect to the model parameters are computed and backpropagated through the network. In contrast to the deterministic autoencoder, the **Variational Autoencoder (VAE)** introduces a probabilistic model to better capture the distribution of the latent variables. A VAE models the data generation process using a latent variable $\mathbf{z} \in \mathbb{R}^l$, and aims to maximize the likelihood of observing the data $\mathbf{x}$ by integrating over all possible latent variables. Specifically, we have the joint distribution:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}), \tag{18.27}$$

where $p(\mathbf{x}|\mathbf{z})$ is the likelihood of the data given the latent variables, and $p(\mathbf{z})$ is the prior distribution of the latent variables, typically chosen to be a standard Gaussian $\mathcal{N}(\mathbf{z}; 0, I)$. The prior assumption that $p(\mathbf{z}) = \mathcal{N}(0, I)$ simplifies the modeling, as it imposes no particular structure on the latent space, which allows for flexible modeling of the data distribution. The encoder in a VAE outputs a distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ over the latent variables, typically modeled as a multivariate Gaussian with mean $\mu_{\theta_e}(\mathbf{x})$ and variance $\sigma_{\theta_e}(\mathbf{x})$, i.e. $q_{\theta_e}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_{\theta_e}(\mathbf{x}), \sigma_{\theta_e}^2(\mathbf{x})I)$. The decoder generates the likelihood of the data $\mathbf{x}$ given the latent variable $\mathbf{z}$, expressed as $p_{\theta_d}(\mathbf{x}|\mathbf{z})$, which typically takes the form of a Gaussian distribution for continuous data. A central challenge in VAE training is the **marginal likelihood** $p(\mathbf{x})$, which represents the probability of the observed data. This marginal likelihood is intractable due to the integral over the latent variables:

$$p(\mathbf{x}) = \int p_{\theta_d}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}. \tag{18.28}$$

To address this, VAE training employs **variational inference**, which approximates the true posterior $p(\mathbf{z}|\mathbf{x})$ with a variational distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$. The goal is to optimize the **Evidence Lower**

Bound (ELBO), which is a lower bound on the log-likelihood $\log p(\mathbf{x})$. The ELBO is derived using **Jensen's inequality**:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_{\theta_e}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta_d}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_{\theta_e}(\mathbf{z}|\mathbf{x})\,||\,p(\mathbf{z})\right), \tag{18.29}$$

where the first term is the expected log-likelihood of the data given the latent variables, and the second term is the **Kullback-Leibler (KL) divergence** between the approximate posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$. The KL divergence acts as a regularizer, penalizing deviations from the prior distribution. The ELBO can then be written as:

$$\mathcal{L}_{\text{VAE}}(\mathbf{x}) = \mathbb{E}_{q_{\theta_e}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta_d}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_{\theta_e}(\mathbf{z}|\mathbf{x})\,||\,p(\mathbf{z})\right). \tag{18.30}$$

This formulation balances two competing objectives: maximizing the likelihood of reconstructing $\mathbf{x}$ from $\mathbf{z}$, and minimizing the divergence between the posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$. In order to perform optimization, we need to compute the gradient of the ELBO with respect to the parameters θ_e and θ_d. However, since sampling from the distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ is non-differentiable, the **reparameterization trick** is applied. This trick allows us to reparameterize the latent variable $\mathbf{z}$ as:

$$\mathbf{z} = \mu_{\theta_e}(\mathbf{x}) + \sigma_{\theta_e}(\mathbf{x}) \cdot \epsilon, \tag{18.31}$$

where $\epsilon \sim \mathcal{N}(0, I)$ is a standard Gaussian noise vector. This enables the backpropagation of gradients through the latent space and allows the optimization process to proceed via stochastic gradient descent. In practice, the **Monte Carlo method** is used to estimate the expectation in the ELBO. This involves drawing K samples $\mathbf{z}_k$ from the variational posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and approximating the expectation as:

$$\hat{\mathcal{L}}_{\text{VAE}}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K}\log p_{\theta_d}(\mathbf{x}|\mathbf{z}_k) - \frac{1}{K}\sum_{k=1}^{K}\log q_{\theta_e}(\mathbf{z}_k|\mathbf{x}). \tag{18.32}$$

This approximation allows for efficient optimization, even when the latent space is high-dimensional and the exact expectation is computationally prohibitive. Thus, the **training process** of a VAE involves the following steps: first, the encoder produces a distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ for each input $\mathbf{x}$; then, latent variables $\mathbf{z}$ are sampled from this distribution; finally, the decoder reconstructs the data $\hat{\mathbf{x}}$ from the latent variable $\mathbf{z}$. The network is trained to maximize the ELBO, which effectively balances the reconstruction loss and the KL divergence term.

In this rigorous exploration, we have presented the mathematical foundations of both autoencoders and variational autoencoders. The core distinction between the two lies in the introduction of a probabilistic framework in the VAE, which leverages variational inference to optimize a tractable lower bound on the marginal likelihood. Through this process, the VAE learns to generate data by sampling from the latent space and reconstructing the input, while maintaining a well-structured latent distribution through regularization by the KL divergence term. The optimization framework for VAEs is grounded in variational inference and the reparameterization trick, enabling gradient-based optimization techniques to efficiently train deep generative models.

18.4 Graph neural networks (GNNs)

Literature Review: Scarselli et. al. (2009) [446] wrote a foundational paper that introduced the concept of Graph Neural Networks (GNNs). It formalized the idea of processing graph-structured data using neural networks, where nodes iteratively update their representations by aggregating information from their neighbors. The paper laid the theoretical groundwork for GNNs, including convergence guarantees and computational frameworks. Kipf and Welling (2017) [447] introduced Graph Convolutional Networks (GCNs), a simplified and highly effective variant of GNNs.

It proposed a localized first-order approximation of spectral graph convolutions, making GNNs scalable and practical for large graphs. GCNs became a cornerstone for many subsequent GNN architectures. Hamilton et. al. (2017) [448] introduced GraphSAGE, a framework for inductive representation learning on large graphs. Unlike transductive methods (e.g., GCN), GraphSAGE generates embeddings for unseen nodes by sampling and aggregating features from a node's local neighborhood. It also introduced mean, LSTM, and pooling aggregators, which are widely used in GNNs. Veličković et. al. (2018) [449] proposed Graph Attention Networks (GATs), which use self-attention mechanisms to compute node representations. GATs assign different weights to neighbors during aggregation, allowing the model to focus on more important nodes. This introduced a new paradigm for handling heterogeneous graph structures. Xu et. al. (2019) [450] analyzed the theoretical expressiveness of GNNs, particularly their ability to distinguish graph structures. It introduced the Graph Isomorphism Network (GIN), which is as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test. The work provided a rigorous framework for understanding the limitations and strengths of GNNs. Gilmer et. al. (2017) [451] formalized the Message Passing Neural Network (MPNN) framework, which generalizes many GNN variants. It introduced a unified view of GNNs as iterative message-passing algorithms, where nodes exchange information with their neighbors. The framework has been widely adopted in molecular and chemical graph analysis. Battaglia et. al. (2018) [452] presented the Graph Networks (GN) framework, which generalizes GNNs to handle relational reasoning over structured data. It introduced a block-based architecture for processing entities, relations, and global attributes, making it applicable to a wide range of tasks, including physics simulations and combinatorial optimization. Bruna et. al. (2014) [453] wrote one of the earliest works that proposed spectral graph convolutions, which use the graph Fourier transform to define convolutional operations on graphs. It laid the foundation for spectral-based GNNs, which later inspired spatial-based methods like GCNs. Ying et. al. (2018) [454] demonstrated the practical application of GNNs in large-scale recommender systems. It introduced PinSage, a GNN-based model that leverages random walks and efficient sampling techniques to handle web-scale graphs. This work highlighted the scalability and real-world impact of GNNs. Zhou et. al. (2020) [455] wrote a comprehensive review paper that summarized the state-of-the-art in GNNs, covering a wide range of methods, applications, and challenges. It provided a taxonomy of GNN architectures, discussed their theoretical foundations, and highlighted open research directions, making it an essential resource for researchers and practitioners.

Graph Neural Networks (GNNs) are a profound and mathematically intricate class of deep learning models specifically designed to handle and process data that is naturally structured as graphs. Unlike traditional neural networks that operate on Euclidean data structures such as vectors, sequences, or grids, GNNs generalize deep learning to non-Euclidean spaces by directly leveraging the underlying graph topology. The mathematical foundation of GNNs is deeply rooted in algebraic graph theory, spectral graph theory, and the principles of geometric deep learning, all of which contribute to the rigorous understanding of how neural networks can be extended to structured relational data. At the core of any graph-based machine learning model lies the mathematical representation of a graph. Formally, a graph G is defined as an ordered pair $G = (V, E)$, where V represents the set of nodes (or vertices), and $E \subseteq V \times V$ represents the set of edges that define the relationships between nodes. The total number of nodes in the graph is denoted by $|V| = N$, while the number of edges is given by $|E|$. The connectivity of the graph is encoded in the adjacency matrix $A \in \mathbb{R}^{N \times N}$, where A_{ij} is nonzero if and only if there exists an edge between nodes i and j. The adjacency matrix can be either binary (indicating the mere presence or absence of an edge) or weighted, in which case A_{ij} encodes the strength or affinity of the connection. In addition to graph connectivity, each node i is often associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, and collecting these feature vectors across all nodes forms the node feature matrix $X \in \mathbb{R}^{N \times d}$, where d is the dimensionality of the feature space.

One of the fundamental challenges in extending neural networks to graph domains is the lack of a consistent node ordering, which makes standard operations such as convolutions, pooling, and fully connected layers non-trivial. Unlike images where a fixed spatial structure allows for well-defined convolutional kernels, graphs exhibit arbitrary structure and permutation invariance, meaning that the labels of nodes can be permuted without altering the intrinsic properties of the graph. This necessitates the development of graph-specific neural network architectures that respect the graph topology while maintaining permutation invariance. To facilitate learning on graphs, GNNs employ a neighborhood aggregation or message-passing scheme, wherein each node iteratively gathers information from its neighbors to update its representation. This process can be formulated mathematically using recursive feature propagation rules. Let $H^{(l)} \in \mathbb{R}^{N \times d_l}$ denote the node feature matrix at layer l, where each row $H_i^{(l)}$ represents the embedding of node i at that layer. The most fundamental form of feature propagation follows the update equation:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{18.33}$$

where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a learnable weight matrix that transforms the feature representation, $\sigma(\cdot)$ is a nonlinear activation function such as ReLU, and $\tilde{A} = A + I$ is the adjacency matrix augmented with self-loops to ensure that each node includes its own features in the aggregation process. The diagonal matrix $\tilde{D}$ is the degree matrix of $\tilde{A}$, defined as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, which normalizes the feature propagation to avoid scale distortion. The initial node features are represented as $H^{(0)} = X$, where X is the matrix of initial node features. The weight matrix at layer l is denoted as $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$, where d_l and d_{l+1} are the dimensions of the input and output feature spaces at layer l, respectively. The weight matrix is trainable. The adjacency matrix A is augmented with self-loops, denoted as $\tilde{A} = A+I$, where I is the identity matrix. The degree matrix $\tilde{D}$ is the diagonal matrix corresponding to the adjacency matrix with self-loops, defined as:

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}, \tag{18.34}$$

where $\tilde{A}_{ij}$ is the entry in the augmented adjacency matrix. The function $\sigma(\cdot)$ is a nonlinear activation function (such as ReLU) applied element-wise to the node features. This operation ensures that each node aggregates information from its local neighborhood, facilitating feature propagation across the graph. More generally, GNNs can be defined using a message passing scheme, which consists of two key steps. Each node i receives messages from its neighbors $j \in N(i)$. The aggregated message at node i at layer l is computed as:

$$m_i^{(l)} = \sum_{j \in N(i)} f_m(H_j^{(l)}, H_i^{(l)}, A_{ij}), \tag{18.35}$$

where f_m is a learnable function that determines how information is aggregated. The node embedding is updated using the function f_u, which takes the current node embedding $H_i^{(l)}$ and the aggregated message $m_i^{(l)}$. The updated embedding for node i at layer $l+1$ is given by:

$$H_i^{(l+1)} = f_u(H_i^{(l)}, m_i^{(l)}), \tag{18.36}$$

where f_u is another learnable function. A popular choice for the functions f_m and f_u is:

$$H_i^{(l+1)} = \sigma\left(W^{(l)} \sum_{j \in N(i)} \tilde{A}_{ij}\tilde{D}_{ii}^{-1}H_j^{(l)}\right), \tag{18.37}$$

where $W^{(l)}$ is the trainable weight matrix, $\tilde{A}_{ij}$ are the entries of the augmented adjacency matrix, and $\tilde{D}_{ii}^{-1}$ is the inverse of the degree matrix. This formulation is permutation invariant, ensuring that the node embeddings do not depend on the order in which neighbors are processed.

A deeper mathematical understanding of GNNs can be obtained by analyzing their connection to spectral graph theory. The Laplacian matrix, central to spectral graph analysis, is defined as

$$L = D - A \tag{18.38}$$

where D is the degree matrix. The normalized Laplacian is given by

$$L_{\text{sym}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{18.39}$$

which possesses an orthonormal eigenbasis. The eigenvalues of the Laplacian encode fundamental properties of the graph, such as connectivity and diffusion characteristics. Spectral methods define graph convolutions in the Fourier domain using the eigen-decomposition

$$L_{\text{sym}} = U \Lambda U^{\top} \tag{18.40}$$

where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. The graph Fourier transform of a signal x is then given by

$$\hat{x} = U^{\top} x \tag{18.41}$$

and graph convolutions are defined as

$$g_{\theta} * x = U g_{\theta}(\Lambda) U^{\top} x \tag{18.42}$$

However, this formulation is computationally expensive, requiring the full eigen-decomposition of L, motivating approximations such as Chebyshev polynomials and first-order simplifications like those used in Graph Convolutional Networks (GCNs). Beyond GCNs, several other variants of GNNs have been developed to address limitations and enhance expressivity. Graph Attention Networks (GATs) introduce an attention mechanism to dynamically weight the contributions of neighboring nodes using learnable attention coefficients. The attention mechanism is formulated as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^{\top}[W h_i \parallel W h_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(a^{\top}[W h_i \parallel W h_k]))} \tag{18.43}$$

where $\parallel$ denotes concatenation, a is a learnable parameter vector, and attention scores α_{ij} determine the importance of neighbors in updating node features. Another variant, GraphSAGE, employs different aggregation functions (mean, LSTM-based, or pooling-based) to sample and aggregate information from local neighborhoods, ensuring scalability to large graphs. The theoretical expressivity of GNNs is an active area of research, particularly in the context of the Weisfeiler-Lehman graph isomorphism test. The Graph Isomorphism Network (GIN) is designed to match the expressiveness of the 1-dimensional Weisfeiler-Lehman test, using an aggregation function of the form:

$$H_i^{(l+1)} = \text{MLP}\left((1+\epsilon) H_i^{(l)} + \sum_{j \in N(i)} H_j^{(l)} \right), \tag{18.44}$$

where $\text{MLP}(\cdot)$ is a multi-layer perceptron, and ϵ is a learnable parameter that controls the contribution of self-information. This formulation has been shown to be more powerful in distinguishing graph structures compared to traditional GCNs. Applications of GNNs span multiple domains, ranging from molecular property prediction in chemistry and biology, where molecules are represented as graphs with atoms as nodes and chemical bonds as edges, to recommendation systems that model users and items as bipartite graphs. Other applications include knowledge graph reasoning, social network analysis, and combinatorial optimization problems.

In summary, Graph Neural Networks represent a mathematically rich extension of deep learning to structured relational data. Their foundation in spectral graph theory, algebraic topology, and geometric deep learning provides a rigorous framework for understanding their function and capabilities. Despite their success, open research challenges remain in improving their expressivity, generalization, and computational efficiency, making them an active and evolving field within modern machine learning.

18.5 Physics Informed Neural Networks (PINNs)

Literature Review: Raissi et. al. (2019) [456] wrote a seminal paper that introduced the foundational framework of PINNs. It demonstrates how neural networks can be trained to solve both forward and inverse problems for nonlinear PDEs by incorporating physical laws (e.g., conservation laws, boundary conditions) directly into the loss function. The authors show the effectiveness of PINNs in solving high-dimensional PDEs, such as the Navier-Stokes equations, and highlight their ability to handle noisy and sparse data. Karniadakis et. al. (2021) [457] wrote a review article that provided a comprehensive overview of physics-informed machine learning, with a focus on PINNs. It discusses the theoretical foundations, challenges, and applications of PINNs in solving PDEs, uncertainty quantification, and data-driven modeling. The paper also highlights the integration of PINNs with other machine learning techniques and their potential for multi-scale and multi-physics problems. Lu et. al. (2021) [458] introduced DeepXDE, a Python library for solving differential equations using deep learning, particularly PINNs. The authors provide a detailed explanation of the library's architecture, its flexibility in handling various types of PDEs, and its ability to solve high-dimensional problems. The paper also includes benchmarks and comparisons with traditional numerical methods. Sirignano and Spiliopoulos (2018) [459] introduced the Deep Galerkin Method (DGM), a precursor to PINNs, which uses deep neural networks to approximate solutions to high-dimensional PDEs. The authors demonstrate the method's effectiveness in solving problems in finance and physics, laying the groundwork for later developments in PINNs. Wang et. al. (2021) [460] addressed a key challenge in training PINNs: the imbalance in gradient flow during optimization, which can lead to poor convergence. The authors propose adaptive weighting schemes and novel architectures to mitigate these issues, significantly improving the robustness and accuracy of PINNs. Mishra and Molinaro (2021) [461] provided a rigorous theoretical analysis of the generalization error of PINNs. The authors derive bounds on the error and discuss the conditions under which PINNs can reliably approximate solutions to PDEs. This paper is crucial for understanding the theoretical limitations and guarantees of PINNs. Zhang et. al. (2020) [462] extended PINNs to solve time-dependent stochastic PDEs by learning in modal space. The authors demonstrate how PINNs can handle uncertainty quantification and stochastic processes, making them applicable to problems in fluid dynamics, materials science, and finance. Jin et. al. (2021) [463] focused on applying PINNs to the incompressible Navier-Stokes equations, a challenging class of PDEs in fluid dynamics. The authors introduce NSFnets, a specialized variant of PINNs, and demonstrate their effectiveness in solving complex flow problems, including turbulent flows. Chen et. al. (2020) [464] showcased the application of PINNs to inverse problems in nano-optics and metamaterials. The authors demonstrate how PINNs can infer material properties and design parameters from limited experimental data, highlighting their potential for real-world engineering applications. Although not explicitly about PINNs, the early work of Psichogios and Ungar (1992) [465] laid the groundwork for integrating physical principles with neural networks. It introduces the concept of hybrid modeling, where neural networks are combined with domain knowledge, a precursor to the physics-informed approach used in PINNs.

Physics Informed Neural Networks (PINNs) are a class of *neural networks* explicitly designed to incorporate *partial differential equations (PDEs)* and *boundary/initial conditions* into their training process. The goal is to find approximate solutions to the PDEs governing physical systems using *neural networks*, while directly embedding the governing *physical laws* (described by PDEs) into the training mechanism.A typical PDE problem is represented as:

$$\mathcal{L}u(x) = f(x), \quad \text{for} \quad x \in \Omega \tag{18.45}$$

where:

- $\mathcal{L}$ is a differential operator, for instance, the *Laplace operator* ∇^2, or the *Navier-Stokes operator* for fluid dynamics.
- $u(x)$ is the unknown solution we wish to approximate.
- $f(x)$ is a known source term, which could represent external forces or other sources in the system.
- Ω is the domain in which the equation is valid, such as a bounded region in $\mathbb{R}^n$ (e.g., $\Omega \subseteq \mathbb{R}^3$).

The solution $u(x)$ is approximated by a neural network $\hat{u}(x, \theta)$, where θ denotes the parameters (weights and biases) of the neural network. A neural network approximates a function $\hat{u}(x)$ as a composition of nonlinear mappings, typically as:

$$\hat{u}(x, \theta) = f_\theta(x) = \sigma(W_k \sigma(W_{k-1} \cdots \sigma(W_1 x + b_1) + b_2) \cdots + b_k) \tag{18.46}$$

where:

- σ is a nonlinear activation function, such as ReLU or sigmoid.
- W_i and b_i are the weight matrices and bias vectors of the i-th layer.
- The function $f_\theta(x)$ is a feedforward neural network with multiple layers.

Thus, the neural network learns a representation $\hat{u}(x, \theta)$ that approximates the physical solution to the PDE. The accuracy of this approximation depends on the choice of the network architecture, activation function, and the training process. To enforce that the neural network approximates a solution to the PDE, we introduce a physics-informed loss function. This loss function typically consists of two parts:

1. *Data-driven loss term*: This term enforces the agreement between the model predictions and any available data points (boundary or initial conditions).
2. *Physics-driven loss term*: This term enforces the satisfaction of the governing PDE at collocation points within the domain Ω.

The data-driven component aims to minimize the discrepancy between the predicted solution and the observed values at certain data points. For a set of training data $\{x_i, u_i\}$, the data-driven loss is given by:

$$\mathcal{L}_{\text{data}} = \sum_{i=1}^{N} |\hat{u}(x_i, \theta) - u_i|^2 \tag{18.47}$$

where $\hat{u}(x_i, \theta)$ is the predicted value of $u(x)$ at x_i, and u_i is the observed value.

The physics-driven term ensures that the predicted solution satisfies the PDE. Let $r(x_i)$ represent the PDE residual evaluated at collocation points $x_i \in \Omega$. The residual $r(x_i)$ is defined as the difference between the left-hand side and the right-hand side of the PDE:

$$r(x_i) = \mathcal{L}\hat{u}(x_i, \theta) - f(x_i) \tag{18.48}$$

Here, $\mathcal{L}\hat{u}(x_i, \theta)$ is the differential operator acting on the neural network approximation $\hat{u}(x_i, \theta)$. By applying *automatic differentiation (AD)*, we can compute the required derivatives of $\hat{u}(x_i, \theta)$ with respect to x. For instance, in the case of a second-order differential equation, AD will compute:

$$\frac{\partial^2 \hat{u}(x)}{\partial x^2} \tag{18.49}$$

The physics-driven loss is then defined as:

$$\mathcal{L}_{\text{physics}} = \sum_{i=1}^{M} r(x_i)^2 \tag{18.50}$$

where $r(x_i)$ represents the residuals at the collocation points x_i distributed throughout the domain Ω. The number of these points M can vary depending on the problem's complexity and dimensionality. The total loss function is a weighted sum of the data-driven and physics-driven terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \mathcal{L}_{\text{physics}} \tag{18.51}$$

where λ is a hyperparameter controlling the balance between the two loss terms. Minimizing this loss function during training ensures that the neural network learns to approximate the solution $u(x)$ that satisfies both the data and the governing physical laws. A key feature of PINNs is the use of *automatic differentiation (AD)*, which allows us to compute the derivatives of the neural network approximation $\hat{u}(x, \theta)$ with respect to its inputs (i.e., the spatial coordinates in the PDE). The chain rule of differentiation is applied recursively through the layers of the neural network.

For a neural network with the following architecture:

$$\hat{u}(x) = f_\theta(x) = \sigma(W_k \sigma(\cdots \sigma(W_1 x + b_1) \cdots + b_k)) \tag{18.52}$$

we can apply *backpropagation* and *automatic differentiation* to compute the derivatives $\frac{\partial \hat{u}(x)}{\partial x}$, $\frac{\partial^2 \hat{u}(x)}{\partial x^2}$, and higher derivatives required by the PDE. For example, for the *Laplace operator* in a 1D setting:

$$\frac{\partial^2 \hat{u}(x)}{\partial x^2} = \sum_{j=1}^{k} W_j \frac{\partial^2 \sigma(\cdot)}{\partial x^2} \tag{18.53}$$

This automatic differentiation procedure ensures that the PDE residual $r(x_i) = \mathcal{L}\hat{u}(x_i, \theta) - f(x_i)$ is computed efficiently and accurately. The formulation of PINNs extends naturally to higher-dimensional PDEs. In the case of a system of partial differential equations, the operator $\mathcal{L}$ may involve higher-order derivatives in multiple dimensions. For instance, in fluid dynamics, the governing equations might involve the *Navier-Stokes equations*, which require computing derivatives in 3D space:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{18.54}$$

Here, $\mathbf{u}(x, t)$ is the velocity field, p is the pressure field, and ν is the kinematic viscosity. The neural network architecture in PINNs can be extended to multi-output networks that solve for vector fields, ensuring that all components of the velocity and pressure fields satisfy the corresponding PDEs. For inverse problems, where we aim to infer unknown parameters of the system (e.g., material properties, boundary conditions), PINNs provide a natural framework. The inverse problem is framed as the optimization of the loss function with respect to both the neural network parameters θ and the unknown physical parameters α:

$$\mathcal{L}_{\text{total}}(\theta, \alpha) = \mathcal{L}_{\text{data}}(\theta, \alpha) + \lambda \mathcal{L}_{\text{physics}}(\theta) \tag{18.55}$$

Multi-fidelity PINNs involve using data at multiple levels of fidelity (e.g., coarse vs. fine simulations, experimental data vs. high-fidelity models) to improve training efficiency and accuracy.

Physics-Informed Neural Networks (PINNs) provide an elegant and powerful approach to solving PDEs by embedding physical laws directly into the training process. The use of automatic differentiation allows for efficient computation of residuals, and the combined loss function enforces both data-driven and physics-driven constraints. With applications spanning across many domains in engineering, physics, and biology, PINNs represent a significant advancement in the integration of machine learning with scientific computing.

18.6 Implementation of the Deep Galerkin Methods (DGM) using the Physics-Informed Neural Networks (PINNs)

Consider the general form of a partial differential equation (PDE) given by:

$$L(u(x)) = f(x), \quad \text{for } x \in \Omega \tag{18.56}$$

where L is a differential operator (either linear or nonlinear), $u(x)$ is the unknown solution, and $f(x)$ is a given source or forcing term. The domain $\Omega \subset \mathbb{R}^d$ represents the spatial region where the solution $u(x)$ is sought. In the case of nonlinear PDEs, L may involve both $u(x)$ and its derivatives in a nonlinear fashion. Additionally, boundary conditions are specified as:

$$u(x) = g(x), \quad x \in \partial\Omega \tag{18.57}$$

where $g(x)$ is a prescribed function at the boundary $\partial\Omega$ of the domain. The weak formulation of the PDE is obtained by multiplying both sides of the differential equation by a test function $v(x)$ and integrating over the domain Ω:

$$\int_\Omega v(x)L(u(x))\,dx = \int_\Omega v(x)f(x)\,dx \tag{18.58}$$

This weak formulation is valid in spaces of functions that satisfy appropriate regularity conditions, such as Sobolev spaces. The weak formulation transforms the problem into an integral form, making it easier to handle numerically. The Deep Galerkin Method (DGM) is a deep learning-based method for approximating the solution of PDEs. The fundamental idea is to construct a neural network-based approximation $\hat{u}(x;\theta)$ for the unknown function $u(x)$, such that the residual of the PDE (the error in satisfying the equation) is minimized in a Galerkin sense. This means that the neural network is trained to minimize the weak form of the PDE's residuals over the domain. In the case of DGM using Physics-Informed Neural Networks (PINNs), the solution is embedded in the architecture of a neural network, and the physics of the problem is enforced through the loss function. The PINN aims to minimize the residual of the weak formulation of the PDE, incorporating both the differential equation and boundary conditions. The neural network used to approximate the solution $\hat{u}(x;\theta)$ is typically a feedforward neural network with an input $x \in \mathbb{R}^d$ (where d is the dimension of the domain) and output $\hat{u}(x;\theta)$, which represents the predicted solution at x. The parameters θ represent the weights and biases of the network, and the architecture is chosen to be deep enough to capture the complexity of the solution. The neural network can be expressed as:

$$\hat{u}(x;\theta) = \text{NN}(x;\theta) \tag{18.59}$$

Here, $\text{NN}(x;\theta)$ denotes the neural network function that maps the input x to an output $\hat{u}(x;\theta)$. The network layers can include nonlinear activation functions such as ReLU or tanh to capture complex behavior. The PINN minimizes a loss function that combines the residual of the PDE and the boundary condition enforcement. Let the loss function be:

$$L(\theta) = L_{\text{PDE}}(\theta) + L_{\text{BC}}(\theta) \tag{18.60}$$

where $L_{\text{PDE}}(\theta)$ represents the loss due to the PDE residual, and $L_{\text{BC}}(\theta)$ enforces the boundary conditions. The PDE residual $L_{\text{PDE}}(\theta)$ is defined as the error in satisfying the PDE at a set of collocation points $\{x_i\}_{i=1}^{N_{\text{coll}}}$ in the domain Ω, where N_{coll} is the number of collocation points. The residual at a point x_i is given by the difference between the differential operator applied to the predicted solution $\hat{u}(x_i;\theta)$ and the forcing term $f(x_i)$:

$$L_{\text{PDE}}(\theta) = \frac{1}{N_{\text{coll}}} \sum_{i=1}^{N_{\text{coll}}} \left(L\left(\hat{u}(x_i;\theta)\right) - f(x_i)\right)^2 \tag{18.61}$$

Here, $L\left(\hat{u}(x_i;\theta)\right)$ is the result of applying the differential operator to the output of the neural network at the collocation point x_i. For nonlinear PDEs, the operator L might involve both $u(x)$ and its derivatives, and the derivatives of $\hat{u}(x;\theta)$ are computed using automatic differentiation. The boundary condition loss $L_{BC}(\theta)$ ensures that the neural network's predictions at boundary points $\{x_{b_i}\}_{i=1}^{N_{BC}}$ satisfy the boundary condition $u(x) = g(x)$. This loss is computed as:

$$L_{BC}(\theta) = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} \left(\hat{u}(x_{b_i};\theta) - g(x_{b_i})\right)^2 \tag{18.62}$$

where x_{b_i} are points on the boundary $\partial\Omega$, and $g(x_{b_i})$ is the prescribed boundary value. For the Training the Neural Network, The objective is to minimize the total loss function:

$$\theta^* = \arg\min_{\theta} \left(L_{PDE}(\theta) + L_{BC}(\theta)\right) \tag{18.63}$$

This minimization is achieved using gradient-based optimization algorithms, such as Stochastic Gradient Descent (SGD) or Adam. The gradients of the loss function with respect to the parameters θ are computed using automatic differentiation, which is a powerful technique in modern deep learning frameworks (e.g., TensorFlow, PyTorch). To achieve a solution in the Galerkin sense, we need to minimize the weak residual of the PDE. The weak residual is derived by integrating the product of the residual and a test function $v(x)$ over the domain:

$$R(x) = L(u(x)) - f(x) \tag{18.64}$$

The weak formulation of the PDE in Galerkin methods ensures that the solution minimizes the projection of the residual onto the space of test functions. In the case of PINNs, the network implicitly constructs this weak form by adjusting the network's parameters to minimize the residual at sampled points. For a general linear PDE, this weak formulation can be expressed as:

$$\int_{\Omega} v(x) \left(L(\hat{u}(x;\theta)) - f(x)\right) \, dx = 0 \tag{18.65}$$

The neural network is designed to minimize the residual $R(x)$ in the weak sense, over the points where the loss is computed.

For nonlinear PDEs, such as the Navier-Stokes equations or nonlinear Schrödinger equations, the neural network's ability to approximate complex functions is key. The operator $L(\hat{u}(x))$ may involve terms like $\hat{u}(x)\nabla\hat{u}(x)$ (nonlinear convection terms), and the neural network can model these nonlinearities by introducing appropriate activation functions in the layers (e.g., ReLU, sigmoid, or tanh). For a nonlinear PDE such as the incompressible Navier-Stokes equations:

$$\frac{\partial \hat{u}}{\partial t} + \hat{u} \cdot \nabla \hat{u} = -\nabla p + \nu \nabla^2 \hat{u} + f \tag{18.66}$$

where $\hat{u}$ is the velocity field, p is the pressure, ν is the kinematic viscosity, and f is the external forcing, the network learns the solution $\hat{u}(x;\theta)$ and $\hat{p}(x;\theta)$, such that:

$$L(\hat{u}(x;\theta), \hat{p}(x;\theta)) = f(x) \tag{18.67}$$

This requires the network to compute the derivatives of $\hat{u}$ and $\hat{p}$ and use them in the residual computation. Collocation points x_i are typically sampled using Monte Carlo methods or Latin hypercube sampling. This allows for efficient exploration of the domain Ω, especially in high-dimensional spaces. Boundary points x_{b_i} are selected to enforce boundary conditions accurately. The training process uses an iterative optimization procedure (e.g., Adam optimizer) to update the neural network parameters θ. The gradients of the loss function are computed using automatic differentiation in deep learning frameworks, ensuring accurate and efficient computation of the derivatives of $\hat{u}(x)$. Convergence is determined by monitoring the reduction in the total loss $L(\theta)$, which should approach zero as the solution is refined. Residuals are monitored for both the PDE and boundary conditions, ensuring that the solution satisfies the PDE and boundary conditions to a high degree of accuracy.

In the Deep Galerkin Method (DGM) using Physics-Informed Neural Networks (PINNs), we construct a neural network to approximate the solution of a PDE in the weak form. The loss function enforces both the PDE residual and boundary conditions, and the network is trained to minimize this loss using gradient-based optimization. The method is highly flexible and can handle both linear and nonlinear PDEs, leveraging the power of neural networks to solve complex differential equations in a scientifically and mathematically rigorous manner. This rigorous framework can be applied to a wide variety of differential equations, from simple linear cases to complex nonlinear systems, and serves as a powerful tool for solving high-dimensional and difficult-to-solve PDEs.

19 Deep Kolmogorov Methods

Literature Review: Han and Jentzen (2017) [479] introduced the Deep BSDE (Backward Stochastic Differential Equation) solver, a foundational framework for solving high-dimensional PDEs using deep learning. It demonstrates how neural networks can approximate solutions to parabolic PDEs by reformulating them as stochastic control problems. The authors rigorously prove the convergence of the method and provide numerical experiments for high-dimensional problems, such as the Hamilton-Jacobi-Bellman equation. Beck et. al. (2021) [480] extended the Deep BSDE method to solve Kolmogorov PDEs, which describe the evolution of probability densities for stochastic processes. The authors provide a theoretical analysis of the approximation capabilities of deep neural networks for these equations and demonstrate the method's effectiveness in high-dimensional settings. While not exclusively focused on Kolmogorov methods, the paper by Raissi et. al. (2019) [456] introduces Physics-Informed Neural Networks (PINNs), which have become a cornerstone in deep learning for PDEs. PINNs incorporate physical laws (e.g., PDEs) directly into the loss function, enabling the solution of forward and inverse problems. The framework is applicable to high-dimensional PDEs and has inspired many subsequent works. Han and Jentzen (2018) [481] provided a comprehensive theoretical and empirical analysis of the Deep BSDE method. It highlights the method's ability to overcome the curse of dimensionality and demonstrates its application to high-dimensional nonlinear PDEs, including those arising in finance and physics. Sirignano and Spiliopoulos (2018) [459] proposed the Deep Galerkin Method (DGM), which uses deep neural networks to approximate solutions to PDEs without requiring a mesh. The method is particularly effective for high-dimensional problems and is shown to outperform traditional numerical methods in certain settings. Yu (2018) [483] introduced the Deep Ritz Method, which uses deep learning to solve variational problems associated with elliptic PDEs. The method is closely related to Kolmogorov methods and provides a powerful alternative for high-dimensional problems. Zhang et. al. (2020) [462] extended PINNs to solve time-dependent stochastic PDEs, including Kolmogorov-type equations. The authors propose a modal decomposition approach to improve the efficiency and accuracy of the method in high dimensions. Jentzen et. al. (2021) [482] provided a rigorous mathematical foundation for deep learning-based methods for nonlinear parabolic PDEs. It includes error estimates and convergence proofs, making it a key reference for understanding the theoretical underpinnings of Deep Kolmogorov Methods. Khoo et. al. (2021) [484] explored the use of neural networks to solve parametric PDEs, which are closely related to Kolmogorov equations. The authors provide a unified framework for handling high-dimensional parameter spaces and demonstrate the method's effectiveness in various applications. While not strictly a deep learning method, the paper by Hutzenthaler et. al. (2020) [485] introduced the Multilevel Picard method, which has inspired many deep learning approaches for high-dimensional PDEs. It provides a theoretical framework for approximating solutions to semilinear parabolic PDEs, including Kolmogorov equations.

The **Deep Kolmogorov Method (DKM)** is a deep learning-based approach to solving high-dimensional partial differential equations (PDEs), particularly those arising from stochastic processes governed by Itô diffusions. The rigorous foundation of DKM is built upon **stochastic analysis, functional analysis, variational principles, and neural network approximation theory**. To fully understand the method, one must rigorously derive the Kolmogorov backward equation, justify its probabilistic representation via Feynman-Kac theory, and establish the error bounds for deep learning approximations within appropriate function spaces. Let us explore these aspects in their maximal mathematical depth.

19.1 The Kolmogorov Backward Equation and Its Functional Formulation

Let X_t be a **d-dimensional Itô diffusion process** defined on a **complete filtered probability space** $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$, satisfying the stochastic differential equation (SDE):

$$dX_t = \mu(X_t)dt + \sigma(X_t)dW_t, \quad X_0 = x, \tag{19.1}$$

where $\mu : \mathbb{R}^d \to \mathbb{R}^d$ is the **drift function** and $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is the **diffusion function**. We assume that both μ and σ satisfy global **Lipschitz continuity** conditions:

$$\|\mu(x) - \mu(y)\| \leq C\|x - y\|, \quad \|\sigma(x) - \sigma(y)\| \leq C\|x - y\|, \quad \forall x, y \in \mathbb{R}^d. \tag{19.2}$$

These conditions guarantee the existence of a **unique strong solution** X_t to the SDE, satisfying $\mathbb{E}[\sup_{0\leq t\leq T} \|X_t\|^2] < \infty$. The **Kolmogorov backward equation** describes the evolution of a function $u(t, x)$, which is defined as the expected value of a terminal function $g(X_T)$ and an integral source term $f(t, X_t)$:

$$u(t, x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s, X_s)ds \mid X_t = x\right]. \tag{19.3}$$

This function satisfies the **parabolic PDE**:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad u(T, x) = g(x), \tag{19.4}$$

where the **second-order differential operator** $\mathcal{L}$, known as the **infinitesimal generator** of X_t, is given by:

$$\mathcal{L}u = \sum_{i=1}^{d} \mu_i(x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{d}(\sigma\sigma^T)_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{19.5}$$

This equation is well-posed in function spaces such as **Sobolev spaces** $H^k(\mathbb{R}^d)$, **Hölder spaces** $C^{k,\alpha}(\mathbb{R}^d)$, or **Bochner spaces** $L^p(\Omega; H^k(\mathbb{R}^d))$ under standard **parabolic regularity** assumptions.

19.2 The Feynman-Kac Representation and Its Justification

To rigorously justify the probabilistic representation of $u(t, x)$, we define the stochastic process:

$$M_t = g(X_T) + \int_t^T f(s, X_s)ds - u(t, X_t). \tag{19.6}$$

Applying **Itô's Lemma** to $u(t, X_t)$, we obtain:

$$dM_t = \left(\frac{\partial u}{\partial t} + \mathcal{L}u - f\right)dt + \nabla u^T \sigma dW_t. \tag{19.7}$$

Since u satisfies the PDE, the drift term vanishes, leaving:

$$dM_t = \nabla u^T \sigma dW_t. \tag{19.8}$$

Taking expectations and noting that the stochastic integral has **zero mean**, we conclude that M_t is a **martingale**, which establishes the **Feynman-Kac representation**:

$$u(t,x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s,X_s)ds \mid X_t = x\right]. \tag{19.9}$$

To prove the above equation, we assume that X_t is a **diffusion process** satisfying the stochastic differential equation (SDE):

$$dX_t = \mu(X_t,t)dt + \sigma(X_t,t)dW_t, \quad X_0 = x. \tag{19.10}$$

Here W_t is a **standard Brownian motion** on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$. The drift $\mu(x,t)$ and diffusion $\sigma(x,t)$ are assumed to be **Lipschitz continuous** in x and **measurable** in t, ensuring **existence and uniqueness** of a strong solution to the SDE. The filtration $(\mathcal{F}_t)$ is the **natural filtration** of W_t, satisfying the **usual conditions** (right-continuity and completeness). We consider the backward **parabolic partial differential equation (PDE)**:

$$\frac{\partial u}{\partial t} + \mu(x,t)\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2(x,t)\frac{\partial^2 u}{\partial x^2} = V(x,t)u + f(x,t), \tag{19.11}$$

with **final condition**:

$$u(x,T) = g(x). \tag{19.12}$$

The Feynman-Kac representation states that:

$$u(x,t) = \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r,r)dr} f(X_s,s)ds + e^{-\int_t^T V(X_r,r)dr} g(X_T) \,\middle|\, X_t = x\right]. \tag{19.13}$$

This provides a **probabilistic representation** of the solution to the PDE. Let's now revisit some prerequisites from Stochastic Calculus and Functional Analysis. For that, we first discuss the existence of the Stochastic Process X_t. The existence of X_t follows from the **standard existence and uniqueness theorem** for SDEs when $\mu(x,t)$ and $\sigma(x,t)$ satisfy the **Lipschitz continuity condition**:

$$|\mu(x,t) - \mu(y,t)| + |\sigma(x,t) - \sigma(y,t)| \leq L|x-y|. \tag{19.14}$$

Under these conditions, there exists a unique **strong solution** X_t that is **adapted** to $\mathcal{F}_t$. Let's use the Itô's Lemma for Stochastic Processes, for a sufficiently smooth function $\phi(X_t,t)$, Itô's lemma states:

$$d\phi(X_t,t) = \left(\frac{\partial \phi}{\partial t} + \mu\frac{\partial \phi}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 \phi}{\partial x^2}\right)dt + \sigma\frac{\partial \phi}{\partial x}dW_t. \tag{19.15}$$

This will be **crucial** in proving the Feynman-Kac formula. Now let us prove the Feynman-Kac Formula. The first step is to define the Stochastic Process Y_s. Define:

$$Y_s = e^{-\int_t^s V(X_r,r)dr}u(X_s,s). \tag{19.16}$$

Applying **Itô's Lemma** to Y_s, we expand:

$$dY_s = d\left(e^{-\int_t^s V(X_r,r)dr}u(X_s,s)\right). \tag{19.17}$$

Using the product rule for stochastic calculus:

$$dY_s = e^{-\int_t^s V(X_r,r)dr}du(X_s,s) + u(X_s,s)d\left(e^{-\int_t^s V(X_r,r)dr}\right) + d\left[e^{-\int_t^s V(X_r,r)dr}, u(X_s,s)\right]. \tag{19.18}$$

Applying **Itô's formula**, we get

$$du(X_s,s) = \left(\frac{\partial u}{\partial t} + \mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2}\right)ds + \sigma\frac{\partial u}{\partial x}dW_s. \tag{19.19}$$

Differentiating the exponential term:

$$d\left(e^{-\int_t^s V(X_r,r)dr}\right) = -V(X_s,s)e^{-\int_t^s V(X_r,r)dr}ds. \tag{19.20}$$

Thus:

$$dY_s = e^{-\int_t^s V(X_r,r)dr}\left(\frac{\partial u}{\partial t} + \mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} - Vu\right)ds + e^{-\int_t^s V(X_r,r)dr}\sigma\frac{\partial u}{\partial x}dW_s. \tag{19.21}$$

The second step shall be taking the expectation and using the Martingale Property. Define the process:

$$M_s = \int_t^s e^{-\int_t^r V(X_q,q)dq}\sigma\frac{\partial u}{\partial x}dW_r. \tag{19.22}$$

Since M_s is a **stochastic integral**, it is a **martingale** with expectation zero:

$$\mathbb{E}[M_T|X_t] = 0. \tag{19.23}$$

Taking expectations on both sides of the equation for Y_s:

$$\mathbb{E}[Y_T|X_t] = Y_t + \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r,r)dr}f ds \,\middle|\, X_t\right]. \tag{19.24}$$

Using the terminal condition $Y_T = e^{-\int_t^T V(X_r,r)dr}g(X_T)$, we obtain:

$$u(x,t) = \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r,r)dr}f(X_s,s)ds + e^{-\int_t^T V(X_r,r)dr}g(X_T) \,\middle|\, X_t = x\right]. \tag{19.25}$$

19.3 Deep Kolmogorov Method: Neural Network Approximation

The **Deep Kolmogorov Method** approximates the function $u(t,x)$ using a deep neural network $u_\theta(t,x)$, parameterized by θ. The loss function is constructed as:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T}\left(u_\theta(t,X_t) - g(X_T) - \int_t^T f(s,X_s)ds\right)^2\right]. \tag{19.26}$$

The parameters θ are optimized via **stochastic gradient descent (SGD)**:

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta\mathcal{L}(\theta_n), \tag{19.27}$$

where η is the learning rate. By the **universal approximation theorem**, a sufficiently deep network with **ReLU activation** satisfies:

$$\|u - u_\theta\|_{L^2} \leq C(L^{-1/2}W^{-1/d}), \tag{19.28}$$

where L is the network depth and W is the network width. Let $u : [0,T]\times\mathbb{R}^d \to \mathbb{R}$ be the exact solution to the Kolmogorov backward equation:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad (t,x)\in[0,T]\times\mathbb{R}^d, \tag{19.29}$$

where $\mathcal{L}$ is the differential operator:

$$\mathcal{L}u = \sum_{i=1}^{d} b_i(x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{d} a_{ij}(x)\frac{\partial^2 u}{\partial x_i\partial x_j}, \tag{19.30}$$

with $b_i(x)$ and $a_{ij}(x)$ satisfying smoothness and uniform ellipticity conditions:

$$\exists\lambda, \Lambda > 0 \quad \text{such that} \quad \lambda|\xi|^2 \leq \sum_{i,j=1}^{d} a_{ij}(x)\xi_i\xi_j \leq \Lambda|\xi|^2 \quad \forall\xi \in \mathbb{R}^d. \tag{19.31}$$

We approximate $u(t, x)$ with a neural network function $u_\theta(t, x)$ of the form:

$$u_\theta(t, x) = \sum_{j=1}^{M} \beta_j\sigma(W_jx + b_j), \tag{19.32}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function, $W_j \in \mathbb{R}^d$, $b_j \in \mathbb{R}$, $\beta_j \in \mathbb{R}$ are trainable parameters, M represents the number of neurons. We seek a bound on the **approximation error**:

$$\|u - u_\theta\|_{H^s}. \tag{19.33}$$

From Sobolev Space Approximation by Deep Neural Networks. We assume $u \in H^s(\mathbb{R}^d)$ with $s > d/2$, so by the **Sobolev embedding theorem**, we obtain:

$$H^s(\mathbb{R}^d) \hookrightarrow C^{0,\alpha}(\mathbb{R}^d), \quad \alpha = s - d/2. \tag{19.34}$$

This ensures u is Hölder continuous, which is crucial for pointwise approximation. From the Universal Approximation in Sobolev Norms, **Barron space theorem** and **error estimates in Sobolev norms**, there exists a neural network u_θ such that:

$$\inf_\theta \|u - u_\theta\|_{H^s} \leq CW^{-s/d}L^{-s/(2d)}. \tag{19.35}$$

where W is the network width,L is the depth, C depends on the smoothness of u. This error bound refines the classical universal approximation theorem by considering **derivatives up to order** s.

To find the Neural Network Approximation Error, let us do the Neural Network Approximation of the Kolmogorov Equation. We now examine the **residual error**:

$$R_\theta(t, x) = \frac{\partial u_\theta}{\partial t} + \mathcal{L}u_\theta - f. \tag{19.36}$$

From Sobolev estimates, we obtain:

$$\|R_\theta\|_{L^2} \leq CW^{-s/d}L^{-s/(2d)}. \tag{19.37}$$

This follows from the **regularity of solutions to parabolic PDEs**, specifically that:

$$\|\mathcal{L}u - \mathcal{L}u_\theta\|_{L^2} \leq C\|u - u_\theta\|_{H^s}. \tag{19.38}$$

Thus, the overall error is:

$$\|u - u_\theta\|_{H^s} \leq CW^{-s/d}L^{-s/(2d)}. \tag{19.39}$$

To find the Asymptotic Rates of Convergence, For large width W and depth L, we analyze the asymptotic behavior:

$$\lim_{W,L\to\infty} \|u - u_\theta\|_{H^s} = 0. \tag{19.40}$$

Moreover, for fixed computational resources, the optimal allocation satisfies:

$$W \sim L^{d/s}. \tag{19.41}$$

This achieves the best rate:

$$\|u - u_\theta\|_{H^s} = \mathcal{O}(L^{-s/(2d)}). \tag{19.42}$$

We have established that the approximation error for deep neural networks in solving the **Kolmogorov backward equation** satisfies the rigorous bound:

$$\|u - u_\theta\|_{H^s} \leq CW^{-s/d} L^{-s/(2d)}, \tag{19.43}$$

which follows from **Sobolev theory, parabolic PDE regularity, and universal approximation in higher-order norms**.

Consider the backward Kolmogorov partial differential equation:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad (t, x) \in [0, T] \times \mathbb{R}^d, \tag{19.44}$$

where the differential operator $\mathcal{L}$ is:

$$\mathcal{L}u = \sum_{i=1}^{d} b_i(x) \frac{\partial u}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^{d} a_{ij}(x) \frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{19.45}$$

By the *Feynman-Kac representation*, the solution is expressed in terms of an expectation over stochastic trajectories:

$$u(t, x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s, X_s) ds \mid X_t = x\right]. \tag{19.46}$$

where X_s follows the *Itô diffusion*:

$$dX_s = b(X_s)ds + \sigma(X_s)dW_s \tag{19.47}$$

for a standard Brownian motion W_s. We approximate this expectation using *Monte Carlo sampling.* Given N independent samples $X_T^{(i)} \sim p(x, T)$, the *empirical Monte Carlo estimator* is:

$$u_N(t, x) = \frac{1}{N} \sum_{i=1}^{N} \left[g(X_T^{(i)}) + \int_t^T f(s, X_s^{(i)}) ds\right]. \tag{19.48}$$

The *Monte Carlo sampling error* is the deviation:

$$E_N = u_N(t, x) - u(t, x). \tag{19.49}$$

For the Measure-Theoretic Representation of Error. Define the probability space:

$$(\Omega, \mathcal{F}, \mathbb{P}), \tag{19.50}$$

where Ω is the sample space of Brownian paths, $\mathcal{F}$ is the filtration generated by W_s, $\mathbb{P}$ is the Wiener measure. The random variable E_N is thus defined over this probability space. By the *Law of Large Numbers (LLN)*, we have

$$\mathbb{P}\left(\lim_{N \to \infty} E_N = 0\right) = 1. \tag{19.51}$$

However, for finite N, we quantify the error using advanced probability bounds. Regarding the Asymptotic Analysis of Monte Carlo Error, the expectation of the squared error is:

$$\mathbb{E}[E_N^2] = \frac{1}{N} \mathrm{Var}\left(g(X_T) + \int_t^T f(s, X_s) ds\right). \tag{19.52}$$

Applying the *Central Limit Theorem (CLT)*, we obtain the asymptotic distribution:

$$\sqrt{N} E_N \xrightarrow{d} \mathcal{N}(0, \sigma^2), \tag{19.53}$$

where:

$$\sigma^2 = \operatorname{Var}\left(g(X_T) + \int_t^T f(s, X_s) ds \right). \tag{19.54}$$

Thus, the Monte Carlo error satisfies:

$$E_N = \mathcal{O}_p \left(\frac{1}{\sqrt{N}} \right). \tag{19.55}$$

We need to find Refined Error Bounds via Concentration Inequalities. To rigorously bound the error, we employ *Hoeffding's inequality*:

$$\mathbb{P} \left(|E_N| \geq \epsilon \right) \leq 2 \exp \left(-\frac{2N\epsilon^2}{\sigma^2} \right). \tag{19.56}$$

For a higher-order bound, we use the *Berry-Esseen theorem*:

$$\sup_x \left| \mathbb{P} \left(\frac{\sqrt{N} E_N}{\sigma} \leq x \right) - \Phi(x) \right| \leq \frac{C}{\sqrt{N}}, \tag{19.57}$$

where C depends on the third moment:

$$\mathbb{E} \left[\left| g(X_T) + \int_t^T f(s, X_s) ds - u(t, x) \right|^3 \right]. \tag{19.58}$$

From a Functional Analysis Perspective, we need to find Operator Norm Bounds. Define the Monte Carlo estimator as a *linear operator*:

$$\mathcal{M}_N : L^2(\Omega) \to \mathbb{R}, \tag{19.59}$$

such that:

$$\mathcal{M}_N \phi = \frac{1}{N} \sum_{i=1}^{N} \phi(X_T^{(i)}). \tag{19.60}$$

The error is then the operator norm deviation:

$$\| \mathcal{M}_N - \mathbb{E} \|_{L^2} = \mathcal{O} \left(\frac{1}{\sqrt{N}} \right). \tag{19.61}$$

By the *spectral decomposition of the covariance operator*, the error satisfies:

$$\| E_N \|_{L^2} \leq \frac{\lambda_{\max}^{1/2}}{\sqrt{N}}, \tag{19.62}$$

where $\lambda_{\max}$ is the largest eigenvalue of the covariance matrix. For a more precise error characterization, we use the *Edgeworth Series* for Higher-Order Expansion:

$$\mathbb{P} \left(\frac{\sqrt{N} E_N}{\sigma} \leq x \right) = \Phi(x) + \frac{\rho_3}{6\sqrt{N}} (1 - x^2) \phi(x) + \mathcal{O} \left(\frac{1}{N} \right), \tag{19.63}$$

where ρ_3 is the skewness of $g(X_T) + \int_t^T f(s, X_s) ds$, $\phi(x)$ is the standard normal density. We have now mathematically rigorously proved that the *Monte Carlo sampling error* in the Deep Kolmogorov method satisfies:

$$E_N = \mathcal{O}_p \left(\frac{1}{\sqrt{N}} \right), \tag{19.64}$$

but with *precise higher-order refinements* via *Berry-Esseen theorem* (finite sample error), *Hoeffding's inequality* (concentration bound), *Functional norm bounds* (operator analysis),*Edgeworth expansion* (higher-order moment corrections). Thus, the optimal error decay rate remains $1/\sqrt{N}$, but the prefactors depend on problem-specific variance and moment conditions.

Therefore, the total approximation error consists of two primary components:

1. **Neural Network Approximation Error**:

$$\|u - u_\theta\|_{L^2} \leq C(L^{-1/2}W^{-1/d}). \tag{19.65}$$

2. **Monte Carlo Sampling Error**:

$$\mathcal{O}(N^{-1/2}), \tag{19.66}$$

   where N is the number of samples used in SGD.

Combining these estimates, we obtain:

$$\|u - u_\theta\|_{L^2} \leq C\left(L^{-1/2}W^{-1/d} + N^{-1/2}\right). \tag{19.67}$$

The **Deep Kolmogorov Method (DKM)** provides a framework for solving high-dimensional PDEs using deep learning, with rigorous theoretical justification from **stochastic calculus, functional analysis, and neural network theory**.

20 Reinforcement Learning

Literature Review: Sutton and Barto (2018) [272] [273] (2021) wrote a definitive textbook on reinforcement learning. It covers the fundamental concepts, including Markov decision processes (MDPs), temporal difference learning, policy gradient methods, and function approximation. The second edition expands on deep reinforcement learning, covering advanced algorithms like DDPG, A3C, and PPO. Bertsekas and Tsitsiklis (1996) [274] laid the theoretical foundation for reinforcement learning by introducing neuro-dynamic programming, an extension of dynamic programming methods for decision-making under uncertainty. It rigorously covers approximate dynamic programming, policy iteration, and value function approximation. Kakade (2003) [275] in his thesis formalized the sample complexity of RL, providing theoretical guarantees for how much data is required for an agent to learn optimal policies. It introduces the PAC-RL (Probably Approximately Correct RL) framework, which has significantly influenced how RL algorithms are evaluated. Szepesvári (2010) [276] presented a rigorous yet concise overview of reinforcement learning algorithms, including value iteration, Q-learning, SARSA, function approximation, and policy gradient methods. It provides deep theoretical insights into convergence proofs and performance bounds. Haarnoja et. al. (2018) [277] introduced Soft Actor-Critic (SAC), an off-policy deep reinforcement learning algorithm that maximizes expected reward and entropy simultaneously. It provides a strong theoretical framework for handling exploration-exploitation trade-offs in high-dimensional continuous action spaces. Mnih et al. (2015) [278] introduced Deep Q-Networks (DQN), demonstrating how deep learning can be combined with Q-learning to achieve human-level performance in Atari games. The authors address key challenges in reinforcement learning, including function approximation and stability improvements. Konda and Tsitsiklis (2003) [279]. provided a rigorous theoretical analysis of Actor-Critic methods, which combine policy-based and value-based learning. It formally establishes convergence proofs for actor-critic algorithms and introduces the natural gradient method for policy improvement. Levine (2018) [280] introduced a probabilistic inference framework for reinforcement learning, linking RL to Bayesian inference. It provides a theoretical foundation for maximum entropy reinforcement learning, explaining why entropy-regularized objectives lead to better exploration and stability. Mannor et. al. (2022) [281] gave one of the most rigorous mathematical treatments of reinforcement learning theory. It covers several topics: PAC guarantees for RL algorithms, Complexity bounds for exploration, Connections between RL and control theory, Convergence rates of popular RL methods. Borkar (2008) [282] rigorously analyzed stochastic approximation methods, which form the theoretical backbone of RL algorithms like TD-learning, Q-learning, and policy gradient methods. Borkar provides a dynamical systems perspective to convergence analysis, offering deep mathematical insights.

20.1 Key Concepts

Reinforcement Learning (RL) is a branch of machine learning that deals with agents making decisions in an environment to maximize cumulative rewards over time. This formalized decision-making process can be described using concepts such as agents, states, actions, and rewards, all of which are mathematically formulated within the framework of a *Markov Decision Process (MDP)*. The following provides an extremely mathematically rigorous discussion of these key concepts. An *agent* interacts with the environment by taking actions based on the current state of the environment. The goal of the agent is to maximize the expected cumulative reward over time. A policy π is a mapping from states to probability distributions over actions. Formally, the policy π can be written as:

$$\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A}), \tag{20.1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{P}(\mathcal{A})$ is the set of probability distributions over the actions. The policy can be either *deterministic*:

$$\pi(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \pi(s_t), \\ 0 & \text{otherwise,} \end{cases} \tag{20.2}$$

where $\pi(s_t)$ is the action chosen in state s_t, or *stochastic*, in which case the policy assigns a probability distribution over actions for each state s_t. The goal of reinforcement learning is to find an *optimal policy* $\pi^*(s_t)$, which maximizes the expected return (cumulative reward) from any initial state. The optimal policy is defined as:

$$\pi^*(s_t) = \arg\max_\pi \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t\right], \tag{20.3}$$

where γ is the *discount factor* that determines the weight of future rewards, and $\mathbb{E}[\cdot]$ represents the expectation under the policy π. The optimal policy can be derived from the *optimal action-value function* $Q^*(s_t, a_t)$, which we define in the next section. The *state* $s_t \in \mathcal{S}$ describes the current situation of the agent at time t, encapsulating all relevant information that influences the agent's decision-making process. The state space $\mathcal{S}$ may be either *discrete* or *continuous*. The state transitions are governed by a probability distribution $P(s_{t+1}|s_t, a_t)$, which represents the probability of moving from state s_t to state s_{t+1} given action a_t. These transitions satisfy the *Markov property*, meaning the future state depends only on the current state and action, not the history of previous states or actions:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t) \quad \forall s_t, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}. \tag{20.4}$$

Additionally, the transition probabilities satisfy the *normalization condition*:

$$\sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) = 1 \quad \forall s_t, a_t. \tag{20.5}$$

The *state distribution* $\rho_t(s_t)$ represents the probability of the agent being in state s_t at time t. The state distribution evolves over time according to the transition probabilities:

$$\rho_{t+k}(s_{t+k}) = \sum_{s_t \in \mathcal{S}} P(s_{t+k}|s_t, a_t)\rho_t(s_t), \tag{20.6}$$

where $\rho_t(s_t)$ is the initial distribution at time t, and $\rho_{t+k}(s_{t+k})$ is the distribution at time $t+k$. An *action* a_t taken at time t by the agent in state s_t leads to a transition to state s_{t+1} and results in a reward r_t. The agent aims to select actions that maximize its long-term reward. The *action-value function* $Q(s_t, a_t)$ quantifies the expected cumulative reward from taking action a_t in state s_t and following the optimal policy thereafter. It is defined as:

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t\right]. \tag{20.7}$$

The optimal action-value function $Q^*(s_t, a_t)$ satisfies the *Bellman Optimality Equation*:

$$Q^*(s_t, a_t) = R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{20.8}$$

This recursive equation provides the foundation for dynamic programming methods such as *value iteration* and *policy iteration*. The optimal policy $\pi^*(s_t)$ is derived by choosing the action that maximizes the action-value function:

$$\pi^*(s_t) = \arg\max_{a_t \in \mathcal{A}} Q^*(s_t, a_t). \tag{20.9}$$

The *optimal value function* $V^*(s_t)$, representing the expected return from state s_t under the optimal policy, is given by:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} Q^*(s_t, a_t). \tag{20.10}$$

The optimal value function satisfies the Bellman equation:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \left[R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \right]. \tag{20.11}$$

The *reward* r_t at time t is a scalar value that represents the immediate benefit (or cost) the agent receives after taking action a_t in state s_t. It is a function $R(s_t, a_t)$ mapping state-action pairs to real numbers:

$$r_t = R(s_t, a_t). \tag{20.12}$$

The agent's objective is to maximize the cumulative reward, which is given by the *total return* from time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{20.13}$$

The agent seeks to find a policy π that maximizes the expected return. The Bellman equation for the expected return is:

$$V^\pi(s_t) = R(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, \pi(s_t)) V^\pi(s_{t+1}). \tag{20.14}$$

This recursive relation helps in solving for the optimal value function. An RL problem is typically modeled as a *Markov Decision Process (MDP)*, which is defined as the tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{20.15}$$

where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $P(s_{t+1}|s_t, a_t)$ is the state transition probability,
- $R(s_t, a_t)$ is the reward function,
- γ is the discount factor.

The agent's goal is to solve the MDP by finding the optimal policy $\pi^*(s_t)$ that maximizes the cumulative expected reward. Reinforcement Learning provides a powerful framework for decision-making in uncertain environments, where the agent seeks to maximize cumulative rewards over time. The core concepts—agents, states, actions, rewards—are formalized mathematically within the structure of a Markov Decision Process, enabling the application of optimization techniques such as dynamic programming, Q-learning, and policy gradient methods to solve complex decision-making problems.

20.2 Deep Q-Learning

Literature Review: Alonso and Arias (2025) [357] rigorously explored the mathematical foundations of Q-learning and its convergence properties. The authors analyze viscosity solutions and the Hamilton-Jacobi-Bellman (HJB) equation, demonstrating how Q-learning approximations align with these principles. The work provides new theoretical guarantees for Q-learning under different function approximation settings. Lu et. al. (2024) [358] proposed a factored empirical Bellman operator to mitigate the curse of dimensionality in Deep Q-learning. The authors provide rigorous theoretical analysis on how factorization reduces complexity while preserving optimality. The study improves the scalability of deep reinforcement learning models. Humayoo (2024) [359] extended Temporal Difference (TD) Learning to deep Q-learning using time-scale separation techniques. It introduces a Q(Δ)-learning approach that improves stability and convergence speed in complex environments. Empirical results validate its performance in Atari benchmarks. Jia et. al. (2024) [360] integrated Deep Q-learning (DQL) with Game Theory for anti-jamming strategies in wireless networks. It provides a rigorous theoretical framework on how multi-agent Q-learning can improve resilience against adversarial attacks. The study introduces multi-armed bandit algorithms and their convergence properties. Chai et. al. (2025) [361] provided a mathematical analysis of transfer learning in non-stationary Markov Decision Processes (MDPs). It extends Deep Q-learning to settings where the environment changes over time, establishing error bounds for Q-learning in these domains. Yao and Gong (2024) [362] developed a resilient Deep Q-network (DQN) model for multi-agent systems (MASs) under Byzantine attacks. The work introduces a novel distributed Q-learning approach with provable robustness against adversarial perturbations. Liu et. al. (2025) [363] introduced SGD-TripleQNet, a multi-Q-learning framework that integrates three Deep Q-networks. The authors provide a mathematical foundation and proof of convergence for their model. The paper bridges reinforcement learning with stochastic gradient descent (SGD) optimization. Masood et. al. (2025) [364] merged Deep Q-learning with Game Theory (GT) to optimize energy efficiency in smart agriculture. It proposes a mathematical model for dynamic energy allocation, proving the existence of Nash equilibria in Q-learning-based decision-making environments. Patrick (2024) [365] bridged economic modeling with Deep Q-learning. It formulates dynamic pricing strategies using deep reinforcement learning (DRL) and provides mathematical proofs on how RL adapts to economic shocks. Mimouni and Avrachenkov (2025) [366] introduced a novel Deep Q-learning algorithm that incorporates the Whittle index, a key concept in optimal stopping problems. It proves convergence bounds and applies the model to email recommender systems, demonstrating improved performance over traditional Q-learning methods.

Deep Q-Learning (DQL) is an advanced reinforcement learning (RL) technique where the goal is to approximate the optimal action-value function $Q^*(s,a)$ through the use of deep neural networks. In traditional Q-learning, the action-value function $Q(s,a)$ maps a state-action pair to the expected return or cumulative discounted reward from that state-action pair, under the assumption of following an optimal policy. Formally, the Q-function is defined as:

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \tag{20.16}$$

where $\gamma \in [0,1]$ is the discount factor, which determines the weight of future rewards relative to immediate rewards, and r_t is the reward received at time step t. The optimal Q-function $Q^*(s,a)$ satisfies the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_0 = s, a_0 = a\right] \tag{20.17}$$

where s_{t+1} is the next state after taking action a in state s, and the maximization term represents the optimal future expected reward. This equation represents the recursive structure of the optimal action-value function, where each Q-value is updated based on the reward obtained in the current step and the maximum future reward expected from the next state. The goal is to learn the optimal Q-function through iterative updates, typically using the Temporal Difference (TD) method. In Deep Q-Learning, the Q-function is approximated by a deep neural network, as directly storing Q-values for every state-action pair is computationally infeasible for large state and action spaces. Let the approximated Q-function be $Q_\theta(s, a)$, where θ denotes the parameters (weights and biases) of the neural network that approximates the action-value function. The deep Q-network (DQN) aims to learn $Q_\theta(s, a)$ such that it closely approximates $Q^*(s, a)$ over time. The update of the Q-function follows the TD error principle, where the goal is to minimize the difference between the current Q-values and the target Q-values derived from the Bellman equation. The loss function for training the DQN is given by:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[(y_t - Q_\theta(s_t, a_t))^2 \right] \tag{20.18}$$

where $\mathcal{D}$ denotes the experience replay buffer containing previous transitions (s_t, a_t, r_t, s_{t+1}). The target y_t for the Q-values is defined as:

$$y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a') \tag{20.19}$$

Here, θ^- represents the parameters of the target network, which is a slowly updated copy of the online network parameters θ. The target network Q_{θ^-} is used to generate stable targets for the Q-value updates, and its parameters are updated periodically by copying the parameters from the online network θ after every T steps. The idea behind this is to stabilize the training by preventing rapid changes in the Q-values due to feedback loops from the Q-network's predictions. The update rule for the network parameters θ follows the gradient descent method and is expressed as:

$$\nabla_\theta L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[(y_t - Q_\theta(s_t, a_t)) \nabla_\theta Q_\theta(s_t, a_t) \right] \tag{20.20}$$

where $\nabla_\theta Q_\theta(s_t, a_t)$ is the gradient of the Q-function with respect to the parameters θ, which is computed using backpropagation through the neural network. This gradient is used to update the parameters of the Q-network to minimize the loss function. In reinforcement learning, the agent must balance exploration (trying new actions) and exploitation (selecting actions that maximize the reward). This is often handled by using an epsilon-greedy policy, where the agent selects a random action with probability ϵ and the action with the highest Q-value with probability $1 - \epsilon$. The epsilon value is decayed over time to ensure that, as the agent learns, it shifts from exploration to more exploitation. The epsilon-greedy action selection rule is given by:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q_\theta(s_t, a), & \text{with probability } 1 - \epsilon \end{cases} \tag{20.21}$$

This policy encourages the agent to explore different actions at the beginning of training and gradually exploit the learned Q-values as training progresses. The decay of ϵ typically follows an annealing schedule to balance exploration and exploitation effectively. A critical component in stabilizing training in Deep Q-Learning is the use of experience replay. In standard Q-learning, the updates are based on consecutive transitions, which can lead to high correlations between consecutive data points. This correlation can slow down learning or even lead to instability. Experience replay addresses this issue by storing a buffer of past experiences and sampling random mini-batches from this buffer during training. This breaks the correlation between consecutive samples and results in more stable and efficient updates. Mathematically, the loss function for training the network involves random sampling of transitions (s_t, a_t, r_t, s_{t+1}) from the experience replay buffer $\mathcal{D}$, and the update to the Q-values is computed using the Bellman error based on the sampled experiences:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[\left(r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a') - Q_\theta(s_t, a_t) \right)^2 \right] \tag{20.22}$$

This method ensures that the Q-values are updated in a way that is less sensitive to the order in which experiences are observed, promoting more stable learning dynamics.

Despite its success, the DQL algorithm can still suffer from certain issues such as overestimation bias and instability due to the maximization step in the Bellman equation. Overestimation bias occurs because the maximization operation $\max_{a'} Q_{\theta^-}(s_{t+1}, a')$ tends to overestimate the true value, as the Q-values are updated based on the same Q-network. To address this, Double Q-learning was introduced, which uses two separate Q-networks for action selection and value estimation, reducing overestimation bias. In Double Q-learning, the target Q-value is computed using the following equation:

$$y_t = r_t + \gamma Q_{\theta^-}\left(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\right) \tag{20.23}$$

This approach helps to mitigate the overestimation problem by decoupling the action selection from the Q-value estimation process. The value of $\arg\max$ is taken from the online network Q_θ, while the Q-value for the next state is estimated using the target network Q_{θ^-}. Another extension to improve the DQL framework is Dueling Q-Learning, which decomposes the Q-function into two separate components: the state value function $V_\theta(s)$ and the advantage function $A_\theta(s, a)$. The Q-function is then expressed as:

$$Q_\theta(s, a) = V_\theta(s) + A_\theta(s, a) \tag{20.24}$$

This decomposition allows the agent to learn the value of a state $V_\theta(s)$ independently of the specific actions, thus reducing the number of parameters needed for learning. This is particularly beneficial in environments where many actions have similar expected rewards, as it enables the agent to focus on identifying the value of states rather than overfitting to individual actions.

In conclusion, Deep Q-Learning is an advanced reinforcement learning method that utilizes deep neural networks to approximate the optimal Q-function, enabling agents to handle large state and action spaces. The mathematical formulation of DQL involves minimizing the loss function based on the temporal difference error, utilizing experience replay to stabilize learning, and using target networks to prevent instability. Extensions such as Double Q-learning and Dueling Q-learning further improve the performance and stability of the algorithm. Despite its remarkable successes, Deep Q-Learning still faces challenges such as overestimation bias and instability, which have been addressed with innovative modifications to the original algorithm.

20.3 Applications in Games and Robotics

Literature Review: Khlifi (2025) [368] applied Double Deep Q-Networks (DDQN) to autonomous driving. The paper discusses the transfer of RL techniques from gaming into self-driving cars, showing how deep RL can handle complex decision-making in dynamic environments. A novel reward function is introduced to improve path efficiency and safety. Kuczkowski (2024) [369] extended multi-objective RL (MORL) to traffic and robotic systems and introduced energy-efficient reinforcement learning for robotics in smart city applications. He also evaluated how RL-based traffic control systems optimize travel time and reduce energy consumption. Krauss et. al. (2025) [370] explored evolutionary algorithms for training RL-based neural networks. The approach integrates mutation-based evolution with reinforcement learning, optimizing RL policies for robot control and gaming AI. This approach shows improvements in learning speed and adaptability in multi-agent robotic environments. Ahamed et. al. (2025) [371] developed RL strategies for robotic soccer, implementing adaptive ball-kicking mechanics and used game engines to train robots, bridging simulated learning and real-world robotics. They also proposed modular robot formations, demonstrating how RL can optimize team play. Elmquist et. al. (2024) [372] focused on sim-to-real transfer in RL for robotics and developed an RL model that can adapt to real-world imperfections (e.g., lighting, texture variations). They used deep learning and image-based metrics to measure differences between simulated and real-world training environments. Kobanda et. al. (2024) [373] introduced a hierarchical approach to offline reinforcement learning (ORL) for robotic control and gaming AI. The study proposes policy subspaces that allow RL models to transfer knowledge across different tasks and demonstrated its effectiveness in goal-conditioned RL for adaptive video game AI. Shefin et. al. (2024) [367] focused on safety-critical RL applications in games and robotic manipulation. They introduced a framework for explainable reinforcement learning (XRL), making AI decisions more interpretable and applied to robotic grasping tasks, ensuring safe and reliable interactions. Xu et. al. (2025) [374] developed UPEGSim, a Gazebo-based simulation framework for underwater robotic games. They used reinforcement learning to optimize evasion strategies in underwater drone combat and highlighted RL applications in military and search-and-rescue robotics. Patadiya et. al. (2024) [375] used Deep RL to create autonomous players in racing games (Forza Horizon 5). They combined AlexNet with DRL for vision-based self-driving agents in gaming. The model learns optimal driving strategies through self-play. Janjua et. al. (2024) [376] explored RL scalability challenges in robotics and open-world games. They studied RL's adaptability in dynamic, open-ended environments (e.g., procedural game worlds) and discussed generalization techniques for RL agents, improving their performance in unpredictable scenarios.

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make decisions by interacting with an environment. The goal of the agent is to maximize a cumulative reward signal over time by taking actions that affect its environment. The RL framework is formally represented by a *Markov Decision Process (MDP)*, which is defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ is the state space, which represents all possible states the agent can be in.
- $\mathcal{A}$ is the action space, which represents all possible actions the agent can take.
- $P(s'|s, a)$ is the state transition probability, which defines the probability of transitioning from state s to state s' under action a.
- $r(s, a)$ is the reward function, which defines the immediate reward received after taking action a in state s.
- $\gamma \in [0, 1)$ is the discount factor, which determines the importance of future rewards.

The objective in RL is for the agent to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes its expected *return* (the cumulative discounted reward), which is mathematically expressed as:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{20.25}$$

where s_t denotes the state at time t, and $a_t = \pi(s_t)$ is the action taken according to the policy π. The expectation is taken over the agent's interaction with the environment, under the policy π. The agent seeks to maximize this expected return by choosing actions that yield the most reward over time. The optimal *value function* $V^*(s)$ is defined as the maximum expected return that can be obtained starting from state s, and is governed by the *Bellman optimality equation*:

$$V^*(s) = \max_a \mathbb{E} \left[r(s, a) + \gamma V^*(s') \right], \tag{20.26}$$

where s' is the next state, and the expectation is taken with respect to the transition dynamics $P(s'|s, a)$. The *action-value function* $Q^*(s, a)$ represents the maximum expected return from taking action a in state s, and then following the optimal policy. It satisfies the Bellman optimality equation for $Q^*(s, a)$:

$$Q^*(s, a) = \mathbb{E} \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right], \tag{20.27}$$

where a' is the next action to be taken, and the expectation is again over the state transition probabilities. These Bellman equations form the basis of many RL algorithms, which iteratively approximate the value functions to learn an optimal policy. To solve these equations, one of the most widely used methods is *Q-learning*, an off-policy, model-free RL algorithm. Q-learning iteratively updates the action-value function $Q(s, a)$ according to the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right], \tag{20.28}$$

where α is the learning rate that controls the step size of updates, and γ is the discount factor. The key idea behind Q-learning is that the agent learns the optimal action-value function $Q^*(s, a)$ without needing a model of the environment. The agent improves its action-value estimates over time by interacting with the environment and receiving feedback (rewards). The iterative nature of this update ensures convergence to the optimal Q^*, under the condition that all state-action pairs are visited infinitely often and α is decayed appropriately. *Policy Gradient* methods, in contrast, directly optimize the policy π_θ, which is parameterized by a vector θ. These methods are useful in high-dimensional or continuous action spaces where action-value methods may struggle. The objective in policy gradient methods is to maximize the expected return, $J(\pi_\theta)$, which is given by:

$$J(\pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{20.29}$$

The policy is updated using the *gradient ascent* method, and the gradient of the expected return with respect to θ is computed as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) Q(s_t, a_t) \right], \tag{20.30}$$

where $Q(s_t, a_t)$ is the action-value function, and $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the *score function*, representing the sensitivity of the policy's likelihood to the policy parameters. By following this gradient, the policy parameters θ are updated to improve the agent's performance. This method, known as *REINFORCE*, is particularly effective when the action space is large or continuous, and the policy needs to be parameterized with complex models, such as deep neural networks. In both Q-learning and policy gradient methods, *exploration* and *exploitation* are essential concepts. *Exploration* refers to trying new actions that have not been sufficiently tested, whereas *exploitation* involves choosing actions that are known to yield high rewards. The *epsilon-greedy* strategy is a common way to balance exploration and exploitation, where with probability ϵ, the agent chooses a random action, and with probability $1 - \epsilon$, it chooses the action with the highest expected reward. As the agent learns, ϵ is typically decayed over time to reduce exploration and focus more on exploiting the learned policy. In more complex environments, *Boltzmann exploration* or *entropy regularization* techniques are used to maintain a controlled amount of randomness in the policy to encourage exploration. In *multi-agent games*, RL takes on additional complexity. When multiple agents interact, the environment is no longer static, as each agent's actions affect the others. In this context, RL can be used to find optimal strategies through *game theory*. A fundamental concept here is the *Nash equilibrium*, where no agent can improve its payoff by changing its strategy, assuming all other agents' strategies remain fixed. In mathematical terms, for two agents i and j, a Nash equilibrium (π_i^*, π_j^*) satisfies:

$$r_i(\pi_i^*, \pi_j^*) \geq r_i(\pi_i, \pi_j^*) \quad \text{and} \quad r_j(\pi_i^*, \pi_j^*) \geq r_j(\pi_i^*, \pi_j), \tag{20.31}$$

where $r_i(\pi_i, \pi_j)$ is the payoff function of agent i when playing policy π_i against agent j's policy π_j. Finding Nash equilibria in multi-agent RL is a complex and computationally challenging task, requiring the agents to learn in a non-stationary environment where the other agents' strategies are also changing over time. In the context of *robotics*, RL is used to solve high-dimensional control tasks, such as *motion planning* and *trajectory optimization*. The robot's state space is often represented by vectors of its position, velocity, and other physical parameters, while the action space consists of control inputs, such as joint torques or linear velocities. In this setting, RL algorithms learn to map states to actions that optimize the robot's performance in a task-specific way, such as minimizing energy consumption or completing a task in the least time. The dynamics of the robot are often modeled by differential equations:

$$\dot{x}(t) = f(x(t), u(t)), \tag{20.32}$$

where $x(t)$ is the state vector at time t, and $u(t)$ is the control input. Through RL, the robot learns to optimize the control policy $u(t)$ to maximize a reward function, typically involving a combination of task success and efficiency. Deep RL, specifically, allows for the representation of highly complex control policies using neural networks, enabling robots to tackle tasks that require high-dimensional sensory input and decision-making, such as object manipulation or autonomous navigation.

In games, RL has revolutionized the field by enabling agents to learn complex strategies in environments where hand-crafted features or simple tabular representations are insufficient. A key challenge in Deep Reinforcement Learning (DRL) is stabilizing the training process, as neural networks are prone to issues such as *overfitting*, *exploding gradients*, and *vanishing gradients*. Techniques such as *experience replay* and *target networks* are used to mitigate these challenges, ensuring stable and efficient learning. Thus, Reinforcement Learning, with its theoretical underpinnings in MDPs, Bellman equations, and policy optimization methods, provides a mathematically rich and deeply rigorous approach to solving sequential decision-making problems. Its application to fields such as games and robotics not only illustrates its versatility but also pushes the boundaries of machine learning into real-world, high-complexity scenarios.

21 Kernel Regression

Literature Review: Fan et. al. (2025) [672] explored kernel regression techniques in causal inference, particularly in the presence of interference among observations. The authors propose an innovative nonparametric estimator that integrates kernel regression with trimming methods, improving robustness in observational studies. Atanasov et. al. (2025) [673] generalized kernel regression by linking it to high-dimensional linear models and stochastic gradient dynamics. The authors present new asymptotics that extend classical results in nonparametric regression and random feature models. Mishra et. al. (2025) [674] applied Gaussian kernel-based regression to image classification and feature extraction. The authors demonstrate how kernel selection significantly impacts model performance in plant leaf detection tasks. Elsayed and Nazier (2025) [675] combined kernel smoothing regression with decomposition analysis to study labor market trends. It highlights the application of kernel-based regression techniques in socio-economic modeling. Kong et. al. (2025) [676] applied Bayesian Kernel Machine Regression (BKMR) to analyze complex relationships between heavy metal exposure and health indicators. It extends kernel regression to toxicology and epidemiological studies. Bracale et. al. (2025) [677] explored antitonic regression methods, establishing new concentration inequalities for regression problems. It highlights kernel methods' superiority over traditional parametric approaches in pricing models. Köhne et. al. (2025) [678] provided a theoretical foundation for kernel regression within Hilbert spaces, focusing on error bounds for kernel approximations in dynamical systems. Sadeghi and Beyeler (2025) [679] applied Gaussian Process Regression (GPR) with a Matérn kernel to estimate perceptual thresholds in retinal implants, showcasing kernel-based regression in biomedical engineering. Naresh et. al. (2025) [680] in a book chapter discussed logistic regression and kernel methods in network security. It illustrates how kernelized models can enhance cybersecurity measures in firewalls. Zhao et. al. (2025) [681] proposed Deep Additive Kernel (DAK) models, which unify kernel methods with deep learning. This approach enhances Bayesian neural networks' interpretability and robustness.

Kernel regression is a non-parametric statistical learning technique that estimates an unknown function $f(x)$ based on a given dataset:

$$\{(x_i, y_i)\}_{i=1}^n, \tag{21.1}$$

where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The fundamental kernel regression estimator is given by:

$$\hat{f}(x) = \sum_{i=1}^n \alpha_i K(x, x_i), \tag{21.2}$$

where $K(x, x')$ is a positive definite kernel function, ensuring that the Gram matrix

$$K = [K(x_i, x_j)]_{i,j=1}^n \tag{21.3}$$

is symmetric positive semi-definite (PSD). The spectral properties of K are crucial for understanding kernel regression's behavior, particularly in the context of regularization, overfitting, and generalization error analysis. To rigorously analyze kernel regression, we consider the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_K$ induced by $K(x, x')$, where functions satisfy:

$$f(x) = \sum_{i=1}^\infty \alpha_i \varphi_i(x), \tag{21.4}$$

where $\varphi_i(x)$ are the eigenfunctions of the integral operator associated with $K(x, x')$:

$$Kf(x) = \int_{\Omega} K(x, x')f(x')d\mu(x'). \tag{21.5}$$

The spectral decomposition of the Kernel Function K takes the form:

$$K\varphi_i = \lambda_i \varphi_i, \quad i = 1, 2, \dots \tag{21.6}$$

where

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq 0 \tag{21.7}$$

are the eigenvalues of K. These eigenvalues and eigenfunctions determine the approximation capacity of kernel regression and its regularization properties.

21.1 Nadaraya–Watson kernel regression

Literature Review: Agua and Bouzebda (2024) [662] explored the Nadaraya–Watson estimator for locally stationary functional time series. It presents asymptotic properties of kernel regression estimators in functional settings, emphasizing how they behave in nonstationary time series. Bouzebda et. al. (2024) [663] generalized Nadaraya–Watson estimators using asymmetric kernels. It rigorously analyzes the Dirichlet kernel estimator and provides first theoretical justifications for its application in conditional U-statistics. Zhao et. al. (2025) [664] applied Nadaraya–Watson regression in engineering applications, specifically in high-voltage circuit breaker degradation modeling. The method smooths interpolated datasets to eliminate measurement errors. Patil et. al. (2024) [665] addressed the bias-variance tradeoff in Nadaraya–Watson kernel regression, showing how optimal smoothing can improve signal denoising and estimation accuracy in noisy environments. Kakani and Radhika (2024) [666] evaluated Nadaraya–Watson estimation in medical data analysis, comparing it with regression trees and other machine learning methods. It highlights the role of bandwidth selection in clinical prediction tasks. Kato (2024) [667] presented a debiased version of Nadaraya–Watson regression, improving its root-N consistency and performance in conditional mean estimation. Sadek and Mohammed (2024) [668] did a comparative study of kernel-based Nadaraya–Watson regression and ordinary least squares (OLS), showing scenarios where nonparametric regression outperforms classical regression techniques. Gong et. al. (2024) [669] introduced Kernel-Thinned Nadaraya–Watson Estimator (KT-NW), which reduces computational cost while maintaining accuracy. This work is highly relevant for large-scale machine learning applications. Zavatone-Veth and Pehlevan (2025) [670] established a theoretical link between Nadaraya–Watson kernel smoothing and statistical physics through the random energy model. It offers new perspectives on kernel regression in high-dimensional settings. Ferrigno (2024) [671] explored how Nadaraya–Watson kernel regression can be applied to reference curve estimation, a key technique in medical statistics and economic forecasting.

Kernel regression is a **non-parametric regression technique** that estimates a function $f(x)$ using a weighted sum of observed values y_i. Given training data $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the Nadaraya–Watson kernel regression estimator takes the form

$$\hat{f}(x) = \frac{\sum_{i=1}^n K_h(x - x_i)y_i}{\sum_{i=1}^n K_h(x - x_i)} \tag{21.8}$$

where $K_h(x)$ is the **scaled kernel function** defined as

$$K_h(x) = \frac{1}{h^d} K\left(\frac{x}{h}\right), \tag{21.9}$$

where h is the bandwidth parameter that determines the smoothing level. A crucial property of kernel functions is their **normalization condition**,

$$\int_{\mathbb{R}^d} K(x)\,dx = 1. \tag{21.10}$$

A common choice for $K(x)$ is the Gaussian kernel:

$$K(x) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\|x\|^2}. \tag{21.11}$$

Let us now do the Bias-Variance Decomposition and Overfitting in Kernel Regression. The performance of kernel regression is governed by the **bias-variance tradeoff**:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \text{Bias}^2 + \text{Variance} + \sigma^2_{\text{noise}}. \tag{21.12}$$

where

$$\text{Bias}(\hat{f}(x)) = \mathbb{E}[\hat{f}(x)] - f(x), \tag{21.13}$$

and

$$\text{Var}(\hat{f}(x)) = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]. \tag{21.14}$$

Expanding $f(x)$ via **Taylor series**, we obtain

$$f(x_i) \approx f(x) + (x_i - x)^T \nabla f(x) + \frac{1}{2}(x_i - x)^T H_f(x)(x_i - x). \tag{21.15}$$

The expectation of the kernel estimate gives

$$\mathbb{E}[\hat{f}(x)] = f(x) + \frac{h^2}{2} \sum_{j=1}^{d} \left(\frac{\int u_j^2 K(u)du}{\int K(u)du} \right) \frac{\partial^2 f}{\partial x_j^2} + O(h^4), \tag{21.16}$$

showing **bias scales as** $O(h^2)$. The variance analysis yields

$$\text{Var}(\hat{f}(x)) \approx \frac{\sigma^2}{nh^d f_X(x)} \int K^2(u)du. \tag{21.17}$$

Thus, variance scales as $O((nh^d)^{-1})$, leading to the **optimal bandwidth selection**

$$h^* \propto n^{-\frac{1}{d+4}}. \tag{21.18}$$

However, when h is too small, overfitting occurs, characterized by high variance:

$$\text{Var}(\hat{f}(x)) \gg 0, \quad \text{Bias}^2(\hat{f}(x)) \approx 0. \tag{21.19}$$

Kernel Ridge Regression (KRR) is one of the best Regularization Techniques to Prevent Overfitting. To control overfitting, we introduce **Tikhonov regularization** in kernel space. Define the **Gram matrix K** with entries

$$K_{ij} = K_h(x_i - x_j). \tag{21.20}$$

We solve the regularized least squares problem:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}. \tag{21.21}$$

The regularization term λ modifies the eigenvalues σ_i of $\mathbf{K}$, giving

$$\alpha_i = \frac{\sigma_i}{\sigma_i + \lambda} v_i^T \mathbf{y}. \tag{21.22}$$

For small λ, inverse eigenvalues σ_i^{-1} amplify noise, whereas for large λ, the regularization term suppresses high-frequency components. In the **spectral decomposition** of $\mathbf{K}$, we write

$$\mathbf{K} = \sum_i \sigma_i v_i v_i^T. \tag{21.23}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the eigenvalues of the kernel matrix $\mathbf{K}$ and $\mathbf{v}_i$ are the orthonormal eigenvectors, i.e.,

$$\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij} \tag{21.24}$$

where δ_{ij} is the Kronecker delta. The rank of $\mathbf{K}$ is equal to the number of nonzero eigenvalues σ_i. The eigenvalues of $\mathbf{K}$ encode the spectrum of feature space correlations. If the kernel function $K(x, x')$ is smooth, the eigenvalues decay exponentially: regularization, the solution is

$$\sigma_i \approx O(i^{-\tau}) \tag{21.25}$$

for some decay exponent $\tau > 0$. The spectral decay controls the effective degrees of freedom of kernel regression. Applying regularization, the solution is

$$\hat{f}(x) = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i + \lambda} v_i^T \mathbf{y} \cdot v_i(x). \tag{21.26}$$

The regularization smoothly filters the high-frequency components of $f(x)$, preventing overfitting. For Controlling Model Complexity in Spectral Filtering, we have to note that large σ_i corresponds to low-frequency components retained in the solution while small σ_i are high-frequency components, attenuated by regularization. The cutoff occurs around defining the effective model complexity. For very small λ,

$$\frac{\sigma_i}{\sigma_i + \lambda} \approx 1, \tag{21.27}$$

causing high variance. For large λ,

$$\frac{\sigma_i}{\sigma_i + \lambda} \approx \frac{\sigma_i}{\lambda}, \tag{21.28}$$

which heavily suppresses small eigenvalues, leading to underfitting. The **optimal** λ is selected via **cross-validation**, minimizing

$$CV(h) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}_{-i}(x_i))^2. \tag{21.29}$$

An alternative approach is smoothing Splines in Kernel Regression which is done by minimizing

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|\mathcal{L} f\|^2, \tag{21.30}$$

where $\mathcal{L}$ is a differential operator like

$$\mathcal{L} = \frac{d^2}{dx^2}. \tag{21.31}$$

This results in the **smoothing spline estimator**

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K_h(x - x_i). \tag{21.32}$$

where $\boldsymbol{\alpha}$ now depends on $\mathbf{K}$ and $\mathcal{L}$. In conclusion, Kernel regression is powerful but prone to overfitting when h is too small, leading to high variance. Regularization techniques such as **kernel ridge regression**, **Tikhonov regularization**, and **smoothing splines** mitigate overfitting by

modifying the spectral properties of the kernel matrix.

There is a Bias-Variance Tradeoff in Spectral Terms. The expected bias measures the deviation of $\hat{f}(x)$ from $f(x)$:

$$\text{Bias}^2 = \sum_{i=1}^{n} (1 - g(\sigma_i))^2 c_i^2. \tag{21.33}$$

where

$$g(\sigma_i) = \frac{\sigma_i}{\sigma_i + \lambda}. \tag{21.34}$$

Large λ shrinks eigenmodes, increasing bias. The variance measures sensitivity to noise:

$$\text{Var} = \sigma^2 \sum_{i=1}^{n} g(\sigma_i)^2. \tag{21.35}$$

For small λ, the model overfits, leading to high variance. The expected generalization bound error in Spectral Terms is given by:

$$E[(\hat{f}(x) - f(x))^2] = \sum_{i} (1 - g(\sigma_i))^2 c_i^2 + \sigma^2 \sum_{i} g(\sigma_i)^2. \tag{21.36}$$

Using asymptotic analysis, the optimal choice of λ is:

$$\lambda^* = O(n^{-\frac{1}{d+4}}). \tag{21.37}$$

which minimizes error and maximizes generalization.

In conclusion, Spectral Properties play an important role in Kernel Regression. The spectral properties of kernel regression determine its ability to generalize and avoid overfitting:

- The eigenvalue decay rate controls approximation power.
- Spectral filtering via regularization prevents high-frequency noise.
- Generalization is optimized when balancing bias and variance.

By leveraging spectral decomposition, we gain a deep understanding of how kernel regression interpolates data while controlling complexity. The optimal choice of λ and h ensures an **optimal tradeoff between bias and variance**, leading to a robust kernel regression model.

21.2 Priestley–Chao kernel estimator

Literature Review: Neumann and Thorarinsdottir (2006) [654] discussed improvements on the Priestley-Chao estimator by addressing its limitations in nonparametric regression. It provides an asymptotic minimax framework for better estimation, particularly in autoregressive models. The modification proposed mitigates the issues arising from bandwidth selection. Steland (2014) [655] applied the Priestley-Chao kernel estimator to stopping rules in time series control charts. This study is significant because it explores its efficiency in dependent data, particularly focusing on bandwidth choice formulas that enhance estimation precision in practical scenarios. Makkulau et. al. (2023) [656] applied the Priestley-Chao estimator in multivariate semiparametric regression. It highlights the estimator's dependence on optimal bandwidth selection and explores modifications that enhance its adaptability in multiple dimensions. Staniswalis (1989) [657] examined the likelihood-based interpretation of kernel estimators and connects the Priestley-Chao approach with generalized maximum likelihood estimation. It rigorously analyzes the estimator's weighting properties and neighborhood selection criteria. Jennen-Steinmetz and Gasser (1988) [652] provided a comparative framework between the Priestley-Chao estimator and other kernel regression estimators. They explore its mathematical properties, convergence rates, and advantages over alternative methods such as Nadaraya-Watson. Mack and Müller (1988) [658] evaluated the Priestley-Chao estimator's error behavior in nonparametric regression. The paper highlights how convolution-type adjustments can improve estimation accuracy under random design conditions. Jones et. al. (2024) [659] categorized various kernel regression estimators, including the Priestley-Chao estimator. It critically evaluates its statistical efficiency and variance properties in comparison to other kernel methods. Ghosh (2015) [660] introduced a variance estimation technique specifically for the Priestley-Chao kernel estimator. The paper presents a method to avoid nuisance parameter estimation, improving computational efficiency. Liu and Luor (2023) [661] integrated fractal interpolants with the Priestley-Chao estimator to handle complex regression problems. It explores modifications to kernel functions that enhance estimation in high-dimensional datasets. Gasser and Muller (1979) [644] wrote a foundational work that revisits the Priestley-Chao estimator in the context of kernel regression. The authors propose two alternative definitions for kernel estimation, aiming to refine the estimator's application in empirical data analysis.

Let $X_1, X_2, \ldots, X_n$ be **independent and identically distributed (i.i.d.)** random variables drawn from an unknown probability density function (PDF) $f(x)$, with cumulative distribution function (CDF) $F(x)$. The goal of nonparametric density estimation is to construct an estimator $\hat{f}(x)$ such that

$$\lim_{n\to\infty} \hat{f}(x) = f(x) \tag{21.38}$$

in some suitable sense, such as pointwise convergence, mean squared error (MSE) consistency, or uniform convergence over compact subsets. In **kernel density estimation (KDE)**, a common approach is to define

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{21.39}$$

where $K(\cdot)$ is a kernel function satisfying

$$\int_{-\infty}^{\infty} K(u)\, du = 1 \tag{21.40}$$

and h is a **bandwidth parameter** controlling the level of smoothing. However, KDE relies on a **fixed bandwidth** h, which can lead to **oversmoothing in regions of high density** and **undersmoothing in regions of low density**. The **Priestley–Chao estimator** improves upon this by **adapting the bandwidth locally**, based on the **spacings between consecutive order statistics**.

There is an important role of Order Statistics and Spacings. Let us define the **order statistics** of the sample as

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)} \tag{21.41}$$

The fundamental insight behind the Priestley–Chao estimator is that **the spacings between order statistics contain direct information about the local density**. Define the **spacing** between two consecutive order statistics as

$$D_i = X_{(i+1)} - X_{(i)}, \quad i = 1, \ldots, n-1 \tag{21.42}$$

Using results from **order statistics theory**, we obtain the key approximation

$$\mathbb{E}[D_i] \approx \frac{1}{n f(X_{(i)})} \tag{21.43}$$

which follows from the fact that the probability of observing a sample in a small interval around $X_{(i)}$ is approximately given by the density $f(X_{(i)})$ times the width of the interval. Thus, rearranging, we obtain the fundamental estimator

$$\hat{f}(X_{(i)}) \approx \frac{1}{nD_i} \tag{21.44}$$

This provides a direct **data-driven way to estimate the density** without choosing a fixed bandwidth h, as in classical KDE methods. Let's now state the formal Definition of the Priestley–Chao Estimator. The **Priestley–Chao kernel estimator** is defined as

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n-1} \frac{1}{D_i} K\left(\frac{x - X_{(i)}}{D_i}\right) \tag{21.45}$$

where $K(\cdot)$ is a **symmetric kernel function** satisfying

$$\int_{-\infty}^{\infty} K(u)\,du = 1, \quad \int_{-\infty}^{\infty} uK(u)\,du = 0, \quad \text{and} \quad \int_{-\infty}^{\infty} u^2K(u)\,du < \infty \tag{21.46}$$

Unlike fixed-bandwidth KDE, here **the bandwidth D_i varies with location**, allowing better adaptation to the underlying density structure. To understand the performance of the estimator, we analyze its **bias and variance**. Using a **first-order Taylor expansion** of D_i around its expectation, we write

$$D_i = \frac{1}{nf(X_{(i)})} + \epsilon_i \tag{21.47}$$

where ϵ_i represents the stochastic deviation from the expected value. Substituting this into the estimator,

$$\hat{f}(X_{(i)}) = \frac{1}{n}\sum_{i=1}^{n-1} \left(f(X_{(i)}) + nf(X_{(i)})^2\epsilon_i\right) K\left(\frac{x - X_{(i)}}{D_i}\right) \tag{21.48}$$

Taking expectations, we obtain the leading-order bias term

$$\mathbb{E}[\hat{f}(x)] = f(x) + \frac{1}{2}h^2 f''(x) + \mathcal{O}(n^{-2/5}) \tag{21.49}$$

where $h = D_i$ represents the **local bandwidth**. The **variance** of the estimator follows from the variance of the spacings, which satisfies

$$\text{Var}[D_i] = \mathcal{O}(n^{-2}) \tag{21.50}$$

Since $\hat{f}(x)$ involves a sum over n terms, its variance is

$$\text{Var}[\hat{f}(x)] = \mathcal{O}(n^{-1}) \tag{21.51}$$

Thus, the **mean squared error (MSE)** is given by

$$\mathbb{E}\left[(\hat{f}(x) - f(x))^2\right] = \text{Bias}^2 + \text{Var} = \mathcal{O}(n^{-4/5}) \tag{21.52}$$

This shows that the **Priestley–Chao estimator achieves the optimal nonparametric rate of convergence**. The kernel function $K(\cdot)$ plays a crucial role in smoothing the estimate. Common choices include:

1. **Uniform kernel**:

$$K(u) = \frac{1}{2}\mathbf{1}(|u| \leq 1) \tag{21.53}$$

2. **Epanechnikov kernel** (optimal in MSE sense):

$$K(u) = \frac{3}{4}(1 - u^2)\mathbf{1}(|u| \leq 1) \tag{21.54}$$

3. **Gaussian kernel**:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2} \tag{21.55}$$

The **integrated squared error (ISE)** is used to optimize kernel selection:

$$\text{ISE} = \int_{-\infty}^{\infty} \left(\hat{f}(x) - f(x) \right)^2 dx \tag{21.56}$$

21.3 Gasser–Müller kernel estimator

Literature Review: Gasser and Muller (1979) [644] wrote one of the foundational papers introducing the Gasser–Müller estimator. It presents the kernel smoothing method and its advantages over existing nonparametric regression techniques, particularly in terms of bias reduction. Gasser and Muller (1984) [645] extended the original estimator to include derivative estimation. It provides rigorous asymptotic analysis of bias and variance, demonstrating the estimator's robustness in various statistical applications. Härdle and Gasser (1985) [646] refined the Gasser–Müller approach by introducing robustness into kernel estimation of derivatives, addressing outlier sensitivity and proposing adaptive bandwidth selection. Müller (1987) [647] generalized the Gasser–Müller estimator by incorporating weighted local regression, improving performance in scenarios with non-uniform data distributions. Chu (1993) [648] proposed an improved version of the Gasser–Müller estimator by modifying its weighting function, leading to better numerical stability and efficiency in practical applications. Peristera and Kostaki (2005) [649] compared various kernel estimators, showing that the Gasser–Müller estimator with a local bandwidth performs better in mortality rate estimation. Müller (1991) [650] addressed the problem of kernel estimators near boundaries, proposing modifications to the Gasser–Müller estimator for improved accuracy at endpoints. Gasser et. al. (2004) [651] expanded on kernel estimation techniques, including the Gasser–Müller estimator, applying them to shape-invariant modeling and structural analysis. Jennen-Steinmetz and Gasser (1988) [652] developed a unified framework for kernel-based estimators, situating the Gasser–Müller approach within a broader nonparametric regression context. Müller (1997) [653] introduced density-adjusted kernel smoothers, improving upon Gasser–Müller estimators in settings with non-uniformly distributed data points.

The **Gasser-Müller kernel estimator** is a sophisticated nonparametric method for estimating the probability density function (PDF) of a continuous random variable. It is an improvement upon the classical kernel density estimator (KDE) and is specifically designed to minimize the boundary bias often present in density estimates near the edges of the sample space. This is achieved by placing the kernel functions at the **midpoints** between adjacent data points rather than directly at the data points themselves. The fundamental modification introduced by Gasser and Müller is crucial for improving the estimator's accuracy in regions close to the boundaries, where traditional kernel density estimation methods tend to perform poorly due to limited data near the boundaries.

Let's now describe the Mathematical Framework of the **Gasser-Müller kernel estimator**. Let $X_1, X_2, \ldots, X_n$ represent a set of n independent and identically distributed (i.i.d.) random variables drawn from an unknown distribution with a probability density function $f(x)$. The goal of kernel density estimation is to estimate this unknown density $f(x)$ based on the observed data. For the Gasser-Müller kernel estimator $\hat{f}_h(x)$, the core idea is to place the kernel function at the midpoint between two consecutive data points, X_i and X_{i+1}, as follows:

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n-1} K_h \left(x - \frac{X_i + X_{i+1}}{2} \right) \tag{21.57}$$

where $\xi_i = \frac{X_i+X_{i+1}}{2}$ is the midpoint between consecutive data points, often referred to as the "midpoint shift", $K_h(x) = \frac{1}{h}K\left(\frac{x}{h}\right)$ is the **scaled kernel function** with bandwidth h, $K(x)$ is the kernel function, typically chosen to be a symmetric probability density, such as the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \tag{21.58}$$

The key difference between the Gasser-Müller estimator and the traditional kernel estimator is the use of midpoints ξ_i instead of the individual data points. The **kernel function** $K_h(x)$ is applied to the **midpoint shift**, effectively smoothing the data and addressing boundary bias by utilizing information from adjacent points.

The **bias** of the estimator can be derived by expanding $\hat{f}_h(x)$ in a Taylor series around the true density $f(x)$. To compute the expected value of $\hat{f}_h(x)$, we first express the expected kernel evaluation:

$$\mathbb{E}[\hat{f}_h(x)] = \frac{1}{n}\sum_{i=1}^{n-1}\mathbb{E}[K_h(x-\xi_i)] \tag{21.59}$$

Since ξ_i is the midpoint of adjacent points X_i and X_{i+1}, we perform a Taylor expansion around the true density $f(x)$, resulting in:

$$\mathbb{E}[\hat{f}_h(x)] = f(x) + \frac{h^2}{2}f''(x)\int_{-\infty}^{\infty} u^2K(u)\,du + O(h^4) \tag{21.60}$$

where $\int_{-\infty}^{\infty} u^2K(u)\,du$ is the **second moment** of the kernel function, denoted σ_K^2. The term $\frac{h^2}{2}f''(x)\sigma_K^2$ represents the **bias** of the estimator, which is quadratic in h. Thus, the bias decreases as h becomes smaller, and for sufficiently smooth densities, this bias is small. The main advantage of the Gasser-Müller method is that it leads to a smaller bias compared to standard kernel density estimators, especially at the boundaries. The **variance** of $\hat{f}_h(x)$ represents the fluctuation of the estimator across different samples. The variance is given by:

$$\mathrm{Var}[\hat{f}_h(x)] = \frac{1}{n}\left(\int_{-\infty}^{\infty} K^2(u)\,du\right)f(x) \tag{21.61}$$

where $\int_{-\infty}^{\infty} K^2(u)\,du$ is the **second moment** of the kernel function $K(x)$. The variance decreases as the sample size n increases, but it also depends on the bandwidth h. For a fixed sample size, the variance is inversely proportional to both h and n, i.e.,

$$\mathrm{Var}[\hat{f}_h(x)] \propto \frac{1}{nh} \tag{21.62}$$

Thus, larger sample sizes and smaller bandwidths lead to smaller variance, but the optimal bandwidth must balance the trade-off between bias and variance. The **mean squared error (MSE)** combines both the bias and the variance to evaluate the overall performance of the estimator. The MSE is given by:

$$\mathrm{MSE}[\hat{f}_h(x)] = \mathrm{Bias}^2 + \mathrm{Var} \tag{21.63}$$

Substituting the expressions for bias and variance, we obtain:

$$\mathrm{MSE}[\hat{f}_h(x)] = \left(\frac{h^2}{2}f''(x)\sigma_K^2\right)^2 + \frac{1}{nh}f(x)\int_{-\infty}^{\infty} K^2(u)\,du \tag{21.64}$$

To minimize the MSE, we select an **optimal bandwidth** h_{opt}. By differentiating the MSE with respect to h and setting the derivative to zero, we obtain the optimal bandwidth that balances the bias and variance:

$$h_{\text{opt}} \propto n^{-1/5}. \tag{21.65}$$

Thus, the optimal bandwidth decreases as the sample size increases, and this scaling behavior is a fundamental characteristic of kernel density estimation.

The **Gasser-Müller estimator** performs exceptionally well when compared to other kernel density estimators, such as the **Parzen-Rosenblatt estimator**. The Parzen-Rosenblatt method places kernels directly at the data points X_i, whereas the Gasser-Müller method places kernels at the midpoints $\xi_i = \frac{X_i+X_{i+1}}{2}$. This simple modification significantly reduces **boundary bias** and results in smoother and more accurate estimates, especially at the boundaries of the sample. Boundary bias occurs in standard KDE methods because kernels at the boundaries have fewer data points to influence them, which leads to a less accurate estimate of the density. Moreover, the Gasser-Müller estimator excels in **derivative estimation**. When estimating the first or second derivatives of the density function, the Gasser-Müller method provides more accurate estimates with lower variance compared to traditional methods. The use of midpoints ensures that the kernel function is better centered relative to the data, reducing boundary effects that are particularly problematic when estimating derivatives. Regarding the Asymptotic Properties, The Gasser-Müller kernel estimator exhibits **asymptotic efficiency**. As the sample size n approaches infinity, the estimator achieves the optimal convergence rate of $O(n^{-1/5})$ for the optimal bandwidth h_{opt}. This convergence rate is the same as that for other kernel density estimators, indicating that the Gasser-Müller estimator is asymptotically efficient. In the limit, the Gasser-Müller estimator is **asymptotically unbiased** and **asymptotically efficient**, meaning that as the sample size increases, the estimator approaches the true density $f(x)$ without bias and with minimal variance. The estimator becomes more accurate as the sample size grows, and the optimal choice of bandwidth ensures that the bias-variance trade-off is well balanced.

In summary, the **Gasser-Müller kernel estimator** offers several distinct advantages over other nonparametric density estimators. Its primary strength lies in its ability to reduce boundary bias by placing kernels at midpoints between adjacent data points. This leads to smoother and more accurate density estimates, especially near the sample boundaries. The optimal choice of bandwidth, which scales as $n^{-1/5}$, balances the bias and variance of the estimator, minimizing the mean squared error. The Gasser-Müller estimator is particularly useful in applications involving density estimation and derivative estimation, where boundary effects and accuracy are crucial. It is a highly effective tool for nonparametric statistical analysis and provides accurate, unbiased estimates even in challenging settings.

21.4 Parzen-Rosenblatt method

Literature Review: Devroye (1992) [557] investigated the efficiency of superkernels in improving the performance of kernel density estimation (KDE). The study introduces higher-order kernels that lead to reduced asymptotic variance without increasing computational complexity. Zambom and Dias (2013) [635] provided a comprehensive review of KDE, discussing its theoretical foundations, bandwidth selection methods, and practical applications in econometrics. The authors emphasize how KDE can outperform traditional histogram methods in economic data analysis. Reyes et. al. (2016) [636] extended KDE to grouped data, proposing a modified Parzen-Rosenblatt estimator for censored and truncated observations. It addresses practical limitations in standard kernel methods when dealing with incomplete datasets. Tenreiro (2024) [637] developed a KDE adaptation for circular data (e.g., angles and periodic phenomena). It provides exact and asymptotic solutions for optimal bandwidth selection in circular KDE. Devroye and Penrod (1984) [638] proved the consistency of automatic KDE methods. It establishes theoretical guarantees on the convergence of density estimates when the bandwidth is chosen through data-driven methods. Machkouri (2011) [639] established asymptotic normality results for KDE when applied to dependent data, particularly strongly mixing random fields. The paper is crucial for extending KDE applications in time series and spatial statistics. Slaoui (2018) [640] introduced a bias reduction technique for KDE, providing theoretical results and practical improvements over the standard Parzen-Rosenblatt estimator. The modifications significantly enhance density estimation in small-sample scenarios. Michalski (2016) [641] used KDE in hydrology to estimate groundwater level distributions. It demonstrates how KDE outperforms parametric methods in environmental science applications. Gramacki and Gramacki (2018) [642] covered KDE fundamentals, implementation details, and computational optimizations. It is an excellent resource for both theoretical insights and practical applications. Desobry et. al. (2007) [643] extended KDE to unordered sets, exploring its use in kernel-based signal processing. It bridges the gap between statistical estimation and machine learning applications.

The **Parzen-Rosenblatt Kernel Density Estimation (KDE) Method** is a foundational technique in non-parametric statistics that allows for the estimation of an unknown probability density function $f(x)$ from a given sample without imposing restrictive parametric assumptions. Mathematically, let $X_1, X_2, \ldots, X_n$ be a set of independent and identically distributed (i.i.d.) random variables drawn from an unknown density $f(x)$. The KDE, which serves as an estimate of $f(x)$, is rigorously defined as

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{21.66}$$

where $K(\cdot)$ is the **kernel function**, and $h > 0$ is the **bandwidth parameter**. The kernel function $K(x)$ serves as a local weighting function that smooths the empirical distribution, while the bandwidth parameter h determines the scale over which the data points contribute to the density estimate. The fundamental goal of KDE is to ensure that $\hat{f}_h(x)$ provides an **asymptotically consistent, unbiased, and efficient** estimator of $f(x)$, all of which require rigorous mathematical conditions to be satisfied. There are some important Properties of the Kernel Function. To ensure the validity of $\hat{f}_h(x)$ as a probability density function estimator, the kernel function $K(x)$ must satisfy the following conditions:

1. **Normalization Condition:**
$$\int_{-\infty}^{\infty} K(x)\, dx = 1 \tag{21.67}$$
This ensures that the kernel behaves like a proper probability density function and does not introduce artificial bias into the estimation.

2. **Symmetry Condition:**
$$K(x) = K(-x), \quad \forall x \in \mathbb{R} \tag{21.68}$$
Symmetry guarantees that the kernel function does not introduce directional bias in the estimation of $f(x)$.

3. **Non-negativity:**
$$K(x) \geq 0, \quad \forall x \in \mathbb{R} \tag{21.69}$$
While not strictly necessary, this property ensures that $\hat{f}_h(x)$ remains a valid probability density estimate in a practical sense.

4. **Finite Second Moment (Variance Condition):**
$$\mu_2(K) = \int_{-\infty}^{\infty} x^2 K(x)\, dx < \infty \tag{21.70}$$
This ensures that the kernel function does not assign an excessive amount of probability mass far from the origin, preserving local smoothness properties.

5. **Unbiasedness Condition (Mean Zero Constraint):**

$$\int_{-\infty}^{\infty} xK(x)\,dx = 0 \tag{21.71}$$

This ensures that the kernel function does not introduce artificial shifts in the density estimate.

Let's discuss the choice of Kernel Function and Examples. Several kernel functions satisfy the above mathematical constraints and are commonly used in KDE:

- **Gaussian Kernel:**

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{21.72}$$

This kernel has the advantage of being infinitely differentiable and providing smooth density estimates.

- **Epanechnikov Kernel:**

$$K(x) = \frac{3}{4}(1 - x^2)\nVdash_{|x|\leq 1} \tag{21.73}$$

This kernel is optimal in the **mean integrated squared error (MISE) sense**, meaning that it minimizes the variance of $\hat{f}_h(x)$ while preserving local smoothness properties.

- **Uniform Kernel:**

$$K(x) = \frac{1}{2}\nVdash_{|x|\leq 1} \tag{21.74}$$

This kernel is simple but suffers from discontinuities, making it less desirable for smooth density estimation.

Regarding the Asymptotic Properties of the KDE, The **bias** of the KDE can be rigorously derived using a second-order Taylor expansion of $f(x)$ around a given evaluation point. Specifically, if $f(x)$ is twice continuously differentiable, we obtain

$$\mathbb{E}[\hat{f}_h(x)] - f(x) = \frac{h^2}{2} f''(x)\mu_2(K) + O(h^4) \tag{21.75}$$

where $\mu_2(K) = \int x^2 K(x)\,dx$ is the second moment of the kernel. The leading term in this expansion shows that the bias is proportional to h^2, implying that a smaller h reduces bias, though at the expense of increasing variance. The **variance** of the KDE is given by

$$\mathrm{Var}[\hat{f}_h(x)] = \frac{1}{nh} f(x) R(K) + O\left(\frac{1}{n}\right) \tag{21.76}$$

where $R(K) = \int K^2(x)\,dx$ measures the roughness of the kernel function. The key observation here is that variance scales as $O(1/(nh))$, implying that a larger h reduces variance but increases bias. To **minimize the mean integrated squared error (MISE)**, one must choose an optimal bandwidth h_{opt} that balances bias and variance. The **optimal bandwidth** is given by

$$h_{\text{opt}} = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{1/5} \tag{21.77}$$

where $\hat{\sigma}$ is the sample standard deviation. This scaling rule, known as **Silverman's rule of thumb**, follows from an asymptotic minimization of

$$\mathbb{E}\left[\int_{-\infty}^{\infty} (\hat{f}_h(x) - f(x))^2\,dx\right] \tag{21.78}$$

which encapsulates both bias and variance effects.

In conclusion, the **Parzen-Rosenblatt method** provides a highly flexible, consistent, and asymptotically optimal approach to density estimation. The choice of **kernel function and bandwidth selection** is critical, as they directly impact the **bias-variance tradeoff**. Future refinements, such as **adaptive bandwidth selection and higher-order kernel corrections**, further enhance its performance.

22 Natural Language Processing (NLP)

Literature Review: Jurafsky and Martin 2023 [225] wrote book that is a cornerstone of NLP theory, covering fundamental concepts like syntax, semantics, and discourse analysis, alongside deep learning approaches to NLP. The book integrates linguistic theory with probabilistic and neural methodologies, making it an essential resource for students and researchers alike. It thoroughly explains sequence labeling, parsing, transformers, and BERT models. Manning and Schütze 1999 [226] wrote a foundational text in NLP, particularly for probabilistic models. It covers hidden Markov models (HMMs), n-gram language models, and expectation-maximization (EM), concepts that still underpin modern transformer-based NLP models. It also introduces latent semantic analysis (LSA), a precursor to modern word embeddings. Liu and Zhang (2018) [227] presented a detailed exploration of deep learning-based NLP, including word embeddings, recurrent neural networks (RNNs), LSTMs, GRUs, and transformers. It introduces the mathematical foundations of neural networks, making it a bridge between classical NLP and deep learning. Allen (1994) [228] wrote a seminal book in NLP, focusing on symbolic and rule-based approaches. It provides detailed coverage of semantic parsing, discourse modeling, and knowledge representation. While it predates deep learning, it forms a strong theoretical foundation for logical and linguistic approaches to NLP. wrote Koehn (2009) [231] wrote a definitive work on statistical NLP, particularly machine translation techniques like phrase-based translation, alignment models, and decoder algorithms. It remains relevant even as neural translation models (e.g., Transformer-based systems) dominate. We now mention some of the recent works in Natural Language Processing (NLP). Hempelmann [230] explored how linguistic theories of humor can be incorporated into Large Language Models (LLMs). It discusses the integration of formal humor theories into neural models and whether LLMs can be used to test linguistic hypotheses. Eisenstein (2020) [232] wrote a modern NLP textbook that bridges theory and practice. It covers both probabilistic and deep learning approaches, including dependency parsing, sequence-to-sequence models, and attention mechanisms. Unlike many texts, it also discusses ethics and bias in NLP models. Otter et. al. (2018) [233] provides a comprehensive review of neural architectures in NLP, covering CNNs, RNNs, attention mechanisms, and reinforcement learning for NLP. It discusses both theoretical implications and empirical advancements, making it an essential reference for deep learning in language tasks. The Oxford Handbook of Computational Linguistics (2022) [234] provides a comprehensive collection of essays covering the entire field of NLP and computational linguistics, including morphology, syntax, semantics, discourse processing, and deep learning applications. It presents theoretical debates and practical applications across different NLP domains. Li et. al. (2025) [229] introduced an advanced multi-head attention mechanism that combines explorative factor analysis with NLP models. It enhances our understanding of how transformers encode syntactic and semantic relationships.

22.1 Text Classification

Literature Review: Liu et. al. (2024) [235] provided a systematic review of text classification techniques, covering traditional machine learning methods (e.g., SVM, Naïve Bayes, Decision Trees) and deep learning approaches (CNNs, RNNs, LSTMs, and transformers). It also discusses feature extraction techniques such as TF-IDF, word embeddings, and BERT-based representations.

Çekik (2025) [236] introduced a rough set-based approach for text classification, highlighting how term weighting strategies impact classification accuracy. It explores feature reduction and entropy-based selection methods to enhance text classifiers. Zhu et. al. (2025) [237] presented a novel entropy-based prefix tuning method for hierarchical text classification. It demonstrates how entropy regularization can enhance transformer-based classifiers like BERT and GPT for multi-label and hierarchical categorization. Matrane et. al. (2024) [238] investigated dialectal text classification challenges in Arabic NLP. It proposes preprocessing optimizations for low-resource dialects and demonstrates how transfer learning improves classification accuracy. Moqbel and Jain (2025) [239] applies text classification to detect deception in online product reviews. It integrates cognitive appraisal theory and NLP-based text mining to distinguish fake vs. genuine reviews. Kumar et. al. (2025) [240] focused on medical text classification, demonstrating how NLP techniques can be applied to diagnose diseases using electronic health records (EHRs) and patient symptoms extracted from text data. Yin (2024) [241] provided a deep dive into aspect-based sentiment analysis (ABSA), discussing challenges in fine-grained text classification. It introduces new BERT-based techniques to improve aspect-level sentiment classification accuracy. Raghavan (2024) [242] examines personality classification using text data. It evaluates the performance of NLP-based personality prediction models and compares lexicon-based, deep learning, and transformer-based approaches. Semeraro et. al. (2025) [243] introduced EmoAtlas, a tool that merges psychological lexicons, artificial intelligence, and network science to perform emotion classification in textual data. It compares its accuracy with BERT and ChatGPT. Cai and Liu (2024) [244] provides a practical approach to text classification in discourse analysis. It explores Python-based techniques for analyzing therapy talk and sentiment classification in conversational texts.

Text classification is a fundamental problem in machine learning and natural language processing (NLP), where the goal is to assign predefined categories to a given text based on its content. This process involves several steps, including text preprocessing, feature extraction, model training, and evaluation. In this answer, we will explore these steps with a focus on the underlying mathematical principles and models used in text classification. The first step in text classification is preprocessing the raw text data. This typically involves the following operations:

- **Tokenization**: Breaking the text into words or tokens.
- **Stopword Removal**: Removing common words (such as ”and”, ”the”, etc.) that do not carry significant meaning.
- **Stemming and Lemmatization**: Reducing words to their base or root form, e.g., ”running” becomes ”run”.
- **Lowercasing**: Converting all words to lowercase to ensure consistency.
- **Punctuation Removal**: Removing punctuation marks.

These operations result in a cleaned and standardized text, ready for feature extraction. Once the text is preprocessed, the next step is to convert the text into numerical representations that can be fed into machine learning models. The most common methods for feature extraction include:

1. Bag-of-Words (BoW) model
2. Term Frequency-Inverse Document Frequency (TF-IDF)

In the first method (**Bag-of-Words (BoW) model**), each document is represented as a vector where each dimension corresponds to a unique word in the corpus. The value of each dimension is the frequency of the word in the document. If we have a corpus of N documents and a vocabulary of M words, the document i can be represented as a vector $\mathbf{x}_i \in \mathbb{R}^M$, where:

$$\mathbf{x}_i = [f(w_1, d_i), f(w_2, d_i), \dots, f(w_M, d_i)] \tag{22.1}$$

where $f(w_j, d_i)$ is the frequency of the word w_j in the document d_i. The BoW model captures only the frequency of terms within the document and disregards their order. While simple and computationally efficient, this model does not capture the syntactic or semantic relationships between words in the document.

A more sophisticated and improved representation can be obtained through Term **Frequency-Inverse Document Frequency (TF-IDF)**, which scales the raw frequency of words by their relative importance in the corpus. TF-IDF is a more advanced technique that aims to weight words based on their importance. It considers both the frequency of a word in a document and the rarity of the word across all documents. The term frequency (TF) of a word w in document d is defined as:

$$\text{TF}(w, d) = \frac{\text{count}(w, d)}{\text{total number of words in } d} \tag{22.2}$$

The inverse document frequency (IDF) is given by:

$$\text{IDF}(w) = \log\left(\frac{N}{\text{DF}(w)}\right) \tag{22.3}$$

where N is the total number of documents and $\text{DF}(w)$ is the number of documents containing the word w. The TF-IDF score is the product of these two:

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \cdot \text{IDF}(w) \tag{22.4}$$

There are several machine learning models that can be used for text classification, ranging from simpler models to more complex ones. A common approach to text classification is to use a linear model such as logistic regression or linear support vector machines (SVM). Given a feature vector $\mathbf{x}_i$ for document i, the prediction of the class label y_i can be made as:

$$\hat{y}_i = \sigma(\mathbf{w}^T \mathbf{x}_i + b) \tag{22.5}$$

where σ is the sigmoid function for binary classification, and $\mathbf{w}$ and b are the weight vector and bias term, respectively. The model parameters $\mathbf{w}$ and b are learned by minimizing a loss function, such as the binary cross-entropy loss. More complex models, such as *Neural Networks (NN)*, involve deeper mathematical formulations. In a typical feedforward neural network, the goal is to learn a set of parameters that map an input vector $\mathbf{x}_i$ to an output label y_i. The network consists of multiple layers of interconnected neurons, each of which applies a non-linear transformation to the input. Given an input vector $\mathbf{x}_i$, the output of the network is computed as:

$$\mathbf{h}_i^{(l)} = \sigma(\mathbf{W}^{(l)} \mathbf{h}_i^{(l-1)} + \mathbf{b}^{(l)}) \tag{22.6}$$

where $\mathbf{h}_i^{(l)}$ is the activation of layer l, σ is the activation function (e.g., ReLU, sigmoid, or tanh), $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias term for layer l. The input to the network is passed through several hidden layers before producing the final classification output. The output layer typically applies a *softmax* function to obtain a probability distribution over the possible classes:

$$P(y_c|\mathbf{x}_i) = \frac{\exp(\mathbf{W}_c^T \mathbf{h}_i + b_c)}{\sum_{c'} \exp(\mathbf{W}_{c'}^T \mathbf{h}_i + b_{c'})} \tag{22.7}$$

where $\mathbf{W}_c$ and b_c are the weights and bias for class c, and $\mathbf{h}_i$ is the output of the last hidden layer. The network is trained by minimizing a *cross-entropy loss function*:

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\sum_{c=1}^{C} y_{i,c} \log P(y_c|\mathbf{x}_i) \tag{22.8}$$

where $y_{i,c}$ is the one-hot encoded label for class c, and the goal is to minimize the difference between the predicted probability distribution and the true class distribution. Throughout the entire process, *optimization* plays a crucial role in fine-tuning model parameters to minimize classification errors. Common optimization techniques include *stochastic gradient descent (SGD)* and its variants, such as *Adam* and *RMSProp*, which update model parameters iteratively based on the gradient of the loss function with respect to the parameters. Given the loss function $\mathcal{L}(\theta)$ parameterized by θ, the gradient of the loss with respect to a parameter θ_i is computed as:

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_i} \tag{22.9}$$

The parameter update rule for gradient descent is then:

$$\theta_i \leftarrow \theta_i - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta_i} \tag{22.10}$$

where η is the learning rate. For each iteration, this update rule adjusts the model parameters in the direction of the negative gradient, ultimately converging to a set of parameters that minimizes the classification error.

In summary, text classification is an advanced and multifaceted problem that requires a deep understanding of various mathematical principles, including linear algebra, probability theory, optimization, and functional analysis. The entire process, from text preprocessing to feature extraction, model training, and evaluation, involves the application of rigorous mathematical techniques that enable the effective classification of text into meaningful categories. Each of these steps, whether simple or complex, plays an integral role in transforming raw text data into actionable insights using mathematically sophisticated models and algorithms.

22.2 Machine Translation

Literature Review: Wu et. al. (2020) [245] introduced end-to-end neural machine translation (NMT), focusing on sequence-to-sequence models, attention mechanisms, and transformer architectures. It explains encoder-decoder frameworks, self-attention, and positional encoding, laying the groundwork for modern NMT. Hettiarachchi et. al. (2024) [246] presented Amharic-to-English machine translation using transformers. It introduces character embeddings and regularization techniques for handling low-resource languages, a critical challenge in multilingual NLP. Das and Sahoo (2024) [247] discussed word alignment models, a fundamental concept in SMT. It explains IBM Model 1-5, HMM alignments, and the role of alignment in phrase-based models. It also explores challenges in handling syntactic divergence across languages. Oluwatoki et. al. (2024) [248] presented one of the first transformer-based Yoruba-to-English MT systems. It highlights how multilingual NLP models struggle with resource-scarce languages and proposes Rouge-based evaluation for MT systems. Uçkan and Kurt [249] discusses the role of word embeddings (Word2Vec, GloVe, FastText) in MT. It covers semantic representation in vector spaces, crucial for context-aware translation in NMT. discussed multiword expressions (MWEs) in MT, a major challenge in NLP. It covers idiomatic expressions, collocations, and phrasal verbs, showing how neural models struggle with multiword disambiguation. Pastor et. al. (2024) [250] discussed multiword expressions (MWEs) in MT, a major challenge in NLP. It covers idiomatic expressions, collocations, and phrasal verbs, showing how neural models struggle with multiword disambiguation. Fernandes (2024) [251] compared open-source large language models (LLMs) and NMT systems in translating spatial semantics in EN-PT-BR (English-Portuguese-Brazilian Portuguese) subtitles. It highlights the limitations of both traditional and neural MT in capturing contextual spatial meanings. Jozić (2024) [252] evaluated ChatGPT's translation capabilities against specialized MT systems like eTranslation (EU Commission MT model). It shows how general-purpose LLMs can rival dedicated NMT systems but struggle with domain-specific translations. Yang (2025) [253] introduced error-detection models for NMT output, using transformer-based classifiers to detect syntactic and semantic errors in machine-generated translations.

Machine Translation (MT) in Natural Language Processing (NLP) is a highly intricate computational task that requires converting text from one language (source language) to another (target language) by using statistical, rule-based, and deep learning models, often underpinned by probabilistic and neural network-based frameworks. The goal is to determine the most probable target sequence $T = \{t_1, t_2, \ldots, t_N\}$ from the given source sequence $S = \{s_1, s_2, \ldots, s_T\}$, by modeling the conditional probability $P(T \mid S)$. The optimal translation is typically defined by:

$$T^* = \arg\max_T P(T \mid S) \tag{22.11}$$

This involves estimating the probability of T given S, with the assumption that the translation can be described probabilistically. In the most fundamental form of statistical machine translation (SMT), this probability is often modeled through a series of translation models that decompose the translation process into manageable components. The conditional probability $P(T \mid S)$ in SMT can be factorized using Bayes' theorem:

$$P(T \mid S) = \frac{P(S, T)}{P(S)} = \frac{P(T \mid S)P(S)}{P(S)} \tag{22.12}$$

Given this decomposition, the core of early SMT models, such as IBM models, sought to model the joint probability $P(S, T)$ over source and target language pairs. Specifically, in word-based models like IBM Model 1, the task reduces to estimating the probability of translating each word in the source language S to its corresponding word in the target language T. The joint probability can be written as:

$$P(S, T) = \prod_{i=1}^{T} \prod_{j=1}^{N} t(s_i \mid t_j) \tag{22.13}$$

where $t(s_i \mid t_j)$ is the probability of translating word s_i in the source sentence to word t_j in the target sentence. The estimation of these probabilities, $t(s_i \mid t_j)$, is typically achieved by analyzing parallel corpora through various techniques such as Expectation-Maximization (EM), which allows the unsupervised learning of these translation probabilities from large amounts of bilingual text data. The EM algorithm iterates between computing the expected alignments of words in the source and target languages and refining the model parameters accordingly. The word-based translation models, however, do not take into account the structure of the language, which often leads to suboptimal translations, especially in languages with significantly different syntactic structures. The challenges stem from the word order differences and idiomatic expressions that cannot be captured through a simple word-to-word mapping. To overcome these limitations, IBM Model 2 introduced the concept of word alignments, where an additional hidden variable A is introduced, representing a possible alignment between words in the source and target sentences. This can be expressed as:

$$P(S, T, A) = \prod_{i=1}^{T} \prod_{j=1}^{N} t(s_i \mid t_j) a(s_i \mid t_j) \tag{22.14}$$

where $a(s_i \mid t_j)$ denotes the alignment probability between word s_i in the source language and word t_j in the target language. By optimizing these alignment probabilities, SMT systems improve the quality of translations by better modeling the relationship between the source and target sentences. Estimating $a(s_i \mid t_j)$, however, requires computationally expensive algorithms, which can be handled by methods like EM for iterative refinement.

A more sophisticated approach was introduced with sequence-to-sequence (Seq2Seq) models, which significantly improved the translation process by leveraging deep learning techniques. The core of Seq2Seq is the encoder-decoder framework, where an encoder processes the entire source sentence and encodes it into a context vector, and a decoder generates the target sequence. In this approach, the translation probability is formulated as:

$$P(T \mid S) = P(t_1 \mid S) \prod_{i=2}^{N} P(t_i \mid t_{<i}, S) \tag{22.15}$$

where $t_{<i}$ denotes the previously generated target words, capturing the sequential nature of translation. The key advantage of the Seq2Seq model is its ability to model entire sentences at once, providing a richer, more flexible representation of both the source and target sequences compared to word-based models. The encoder, typically implemented using Recurrent Neural Networks (RNNs) or more advanced variants such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks, encodes the source sequence S into hidden states. The hidden state at time step t is computed recursively, based on the input x_t (the source word representation at time step t) and the previous hidden state h_{t-1}:

$$h_t = f(h_{t-1}, x_t) \tag{22.16}$$

where f represents the update function, which is often parameterized as a non-linear function, such as a sigmoid or tanh. This recursion generates a sequence of hidden states $\{h_1, h_2, \ldots, h_T\}$, each encoding the relevant information of the source sentence. In this model, the decoder generates the target sequence one token at a time by conditioning on the previous tokens $t_{<i}$ and the context vector c, which is typically the last hidden state from the encoder. The conditional probability of generating the next target word is given by:

$$P(t_i \mid t_{<i}, S) = \text{softmax}(W h_t) \tag{22.17}$$

where W is a learned weight matrix, and h_t is the hidden state of the decoder at time step t. The softmax function converts the output of the network into a probability distribution over the vocabulary, and the word with the highest probability is chosen as the next target word.

A significant improvement to Seq2Seq was introduced through the attention mechanism. This allows the decoder to dynamically focus on different parts of the source sentence during translation, instead of relying on a single fixed-length context vector. The attention mechanism computes a set of attention weights $\alpha_{t,i}$ for each source word, which are used to compute a weighted sum of the encoder's hidden states to form a dynamic context vector c_t. The attention weight $\alpha_{t,i}$ for time step t in the decoder and source word i is calculated as:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{k=1}^{T} \exp(e_{t,k})} \tag{22.18}$$

where $e_{t,i} = \text{score}(h_t, h_i)$ is a learned scoring function, which can be modeled as:

$$e_{t,i} = \vec{v}^{\top} \tanh(W_1 h_t + W_2 h_i) \tag{22.19}$$

This attention mechanism allows the model to adaptively focus on relevant parts of the source sentence while generating each word in the target sentence, thus overcoming the limitations of fixed-length context vectors in long sentences. Training a machine translation model typically involves optimizing a loss function that quantifies the difference between the predicted target sequence and the true target sequence. The most common loss function is the negative log-likelihood:

$$L(\theta) = -\sum_{i=1}^{N} \log P(t_i \mid t_{<i}, S; \theta) \tag{22.20}$$

where θ represents the parameters of the model. The parameters of the neural network are updated using gradient-based optimization techniques, such as stochastic gradient descent (SGD) or Adam, with the gradient of the loss function with respect to each parameter being computed via backpropagation. In backpropagation, the gradient is computed by recursively applying the chain rule through the layers of the network. For a parameter θ, the gradient is given by:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial L(\theta)}{\partial y} \frac{\partial y}{\partial \theta} \tag{22.21}$$

where y represents the output of the network, and $\frac{\partial L(\theta)}{\partial y}$ is the gradient of the loss with respect to the output. These gradients are then propagated backward through the network to update the parameters, thereby minimizing the loss function. The quality of a translation is often evaluated using automatic metrics such as BLEU (Bilingual Evaluation Understudy), which measures the n-gram overlap between the machine-generated translation and human references. The BLEU score for an n-gram of length n is computed as:

$$\text{BLEU}(T, R) = \exp\left(\sum_{n=1}^{N} w_n \log p_n(T, R)\right) \tag{22.22}$$

where $p_n(T, R)$ is the precision of n-grams between the target translation T and reference R, and w_n is the weight assigned to each n-gram length. Despite advancements, machine translation still faces challenges, such as handling rare or out-of-vocabulary words, idiomatic expressions, and the alignment of complex syntactic structures across languages. Approaches such as transfer learning, unsupervised learning, and domain adaptation are being explored to address these issues and improve the robustness and accuracy of MT systems.

22.3 Chatbots and Conversational AI

Literature Review: Linnemann and Reimann (2024) [254] explored how conversational AI, particularly chatbots, affects human interactions and social psychology. It discusses the role of Large Language Models (LLMs) and their applications in dialogue systems, providing a theoretical perspective on chatbot integration into human communication. Merkel and Schorr (2024) [255] categorizes different types of conversational agents and their NLP capabilities. It discusses the evolution from rule-based chatbots to transformer-based models, emphasizing how natural language processing has enhanced chatbot usability. Kushwaha and Singh (2022) [256] provided a technical analysis of chatbot architectures, covering intent recognition, entity extraction, and dialogue management. It compares traditional ML-based chatbot models with deep learning approaches. Macedo et. al. (2024) [257] presented a healthcare-oriented chatbot that leverages conversational AI to assist Parkinson's patients. It details speech-to-text and NLP techniques used for interactive healthcare applications. Gupta et. al. (2024) [258] outlines the theoretical foundations of generative AI-based chatbots, explaining how LLMs like ChatGPT influence conversational AI. It also introduces a framework for evaluating chatbot effectiveness. Foroughi and Iranmanesh (2025) [259] examined how AI-powered chatbots influence consumer behavior in e-commerce. It introduces a theoretical framework to understand chatbot adoption and trust. Jandhyala (2024) [260] provided a deep dive into chatbot development, covering NLP techniques, intent recognition, and multi-turn dialogue management. It also discusses best practices for chatbot deployment. Pavlović and Savić (2024) [261] explored the use of conversational AI in digital marketing, analyzing how LLM-based chatbots improve customer experience. It also evaluates sentiment analysis and feedback loops in chatbot interactions. Mannava et. al. (2024) [262] examined the ethical and functional aspects of chatbots in child education, focusing on how NLP models must be adjusted for child-appropriate interactions. Sherstinova and Mikhaylovskiy (2024) [263] focused on language-specific challenges in chatbot NLP, discussing how conversational AI models struggle with morphologically rich languages like Russian.

Chatbots and Conversational AI have evolved as some of the most sophisticated applications of Natural Language Processing (NLP), a subfield of artificial intelligence that strives to enable machines to understand, generate, and interact in human language. At the core of conversational AI is the ability to generate meaningful, contextually appropriate responses in a coherent and fluent manner. This challenge is deeply rooted in both the complexities of natural language itself and the mathematical models that attempt to approximate human understanding. This intricate task involves processing language at different levels: syntactic (structure), semantic (meaning), and pragmatic (context). These systems employ probabilistic and algebraic techniques to handle language complexities and employ *statistical models*, *deep neural networks*, and *optimization algorithms* to generate, understand, and respond to language.

In mathematical terms, conversational AI can be seen as a sequence of transformations from one set of words or symbols (the input) to another (the output). The first mathematical aspect is *language modeling*, which is crucial for predicting the likelihood of word sequences. The probability distribution of a sequence of words $w_1, w_2, \ldots, w_n$ is generally computed using the chain rule of probability:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{22.23}$$

where $P(w_i | w_1, w_2, \ldots, w_{i-1})$ models the conditional probability of the word w_i given all the preceding words. This is a central concept in language generation tasks. In traditional *n-gram models*, this conditional probability is estimated by considering only a fixed number of previous words. The *bigram* model, for instance, assumes that the probability of a word depends only on the previous word, leading to:

$$P(w_i | w_1, w_2, \ldots, w_{i-1}) \approx P(w_i | w_{i-1}) \tag{22.24}$$

However, more advanced conversational AI systems, such as those based on *recurrent neural networks (RNNs)*, attempt to model dependencies over much longer sequences. RNNs, in particular, process the input sequence $w_1, w_2, \ldots, w_n$ recursively by maintaining a hidden state h_t that captures the context up to time t. The hidden state is computed by:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{22.25}$$

where σ is a non-linear activation function (e.g., *tanh* or *sigmoid*), W_h, W_x are weight matrices, and b is a bias term. While RNNs provide a mechanism to capture sequential dependencies, they suffer from the *vanishing gradient* problem, particularly for long sequences. To address this issue, *Long Short-Term Memory (LSTM)* units and *Gated Recurrent Units (GRUs)* were introduced, with special gating mechanisms that help mitigate the loss of information over long time horizons. These networks introduce memory cells and gates, which regulate the flow of information in the network. For instance, the LSTM memory cell is governed by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{22.26}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad h_t = o_t \cdot \tanh(c_t) \tag{22.27}$$

where f_t, i_t, o_t are the forget, input, and output gates, respectively, and c_t represents the cell state, which carries information across time steps. The LSTM thus enables better capture of long-range dependencies by controlling the flow of information in a more structured way. In more recent times, *transformer models* have revolutionized conversational AI by replacing the sequential nature of RNNs with parallelized self-attention mechanisms. The transformer model uses *multi-head self-attention* to weigh the importance of each word in a sequence relative to all other words. The self-attention mechanism computes a weighted sum of values V based on queries Q and keys K, with the attention being computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{22.28}$$

where d_k is the dimension of the key vectors. This operation allows the model to attend to all parts of the input sequence simultaneously, enabling better handling of long-range dependencies and improving computational efficiency by processing sequences in parallel. Unlike RNNs, transformers do not process tokens in a fixed order but instead utilize positional encoding to inject sequence order information. The positional encoding for position i and dimension $2k$ is given by:

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{2k/d}}\right), \quad PE(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d}}\right) \tag{22.29}$$

where d is the embedding dimension and k is the index for the dimension of the positional encoding. This approach allows transformers to handle longer sequences more efficiently than RNNs and LSTMs, and is the basis for models like *BERT*, *GPT*, and other state-of-the-art conversational models. Semantic understanding in conversational AI involves translating sentences into formal representations that can be manipulated by the system. A well-known approach for capturing meaning is *compositional semantics*, which treats the meaning of a sentence as a function of the meanings of its parts. For this, *lambda calculus* is often employed to represent the meaning of sentences as functions that operate on their arguments. For example, the sentence "John saw the car" can be represented as a lambda expression:

$$\lambda x.\text{see}(x, \text{car}) \tag{22.30}$$

where $\text{see}(x, y)$ is a predicate representing the action of seeing, and λx quantifies over the subject of the action. This allows for the compositional building of complex meanings from simpler components. Dialogue management is another critical aspect of conversational AI systems. This is the process of maintaining coherence and context over the course of a conversation. It involves understanding the user's input in light of prior dialogue history and generating a response that is contextually relevant. To model the dialogue state, *Markov Decision Processes (MDPs)* are commonly employed. In this context, the dialogue state is represented as a set of possible states, with actions being transitions between these states. The goal is to select actions (responses) that maximize cumulative rewards, which, in this case, corresponds to maintaining a coherent and engaging conversation. The value function $V(s)$ at state s can be computed using the Bellman equation:

$$V(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s') \right] \tag{22.31}$$

where $R(s, a)$ is the immediate reward for taking action a from state s, γ is the discount factor, and $P(s'|s, a)$ represents the transition probability to the next state s' given action a. By solving this equation, the system can determine the optimal policy for responding to user inputs in a way that maximizes long-term conversational quality. Once the dialogue state is updated, the next step in conversational AI is to generate a response. This is typically achieved using *sequence-to-sequence models*, in which the input sequence (e.g., the user's query) is processed by an encoder to produce a fixed-size context vector, and a decoder generates the output sequence (e.g., the chatbot's response). The basic structure of these models can be expressed as:

$$y_t = \text{Decoder}(y_{t-1}, h_t) \tag{22.32}$$

where y_t represents the token generated at time t, and h_t is the hidden state passed from the encoder. Attention mechanisms are incorporated into this framework to allow the decoder to focus on different parts of the input sequence at each step, improving the quality of the generated response. Training conversational models requires optimizing parameters through *backpropagation* and *gradient descent*. The loss function, typically *cross-entropy loss*, is minimized to update the model's parameters:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{22.33}$$

where $\hat{y}_i$ is the predicted probability for the correct token y_i, and N is the length of the sequence. The parameters θ are updated iteratively through gradient descent, adjusting the weights to minimize the error.

In summary, chatbots and conversational AI systems are grounded in a rich mathematical framework involving statistics, linear algebra, optimization, and neural networks. Each step, from language modeling to dialogue management, relies on carefully constructed mathematical foundations that drive the ability of machines to interact intelligently and meaningfully with humans. Through advancements in deep learning and optimization techniques, conversational AI continues to push the boundaries of what machines can understand and generate in natural language, leading to more sophisticated, human-like interactions.

23 Deep Learning Frameworks

23.1 TensorFlow

Literature Review: Takhsha et. al. (2025) [283] introduced a TensorFlow-based framework for medical deep learning applications. The authors propose a novel deep learning diagnostic system that integrates Choquet integral theory with TensorFlow-based models, improving the explainability of deep learning decisions in medical imaging. Singh and Raman (2025) [284] extended TensorFlow to Graph Neural Networks (GNNs), discussing how TensorFlow's computational graph structure aligns with graph theory. It provides a rigorous mathematical foundation for applying deep learning to non-Euclidean data structures. Yao et. al. (2024) [285] critically analyzed TensorFlow's vulnerabilities to adversarial attacks and introduces a robust deep learning ensemble framework. The authors explore autoencoder-based anomaly detection using TensorFlow to enhance cybersecurity defenses. Chen et. al. (2024) [286] provided an extensive comparison of TensorFlow pretrained models for various big data applications. It discusses techniques like transfer learning, fine-tuning, and self-supervised learning, emphasizing how TensorFlow automates hyperparameter tuning. Dumić (2024) [287] wrote as a rigorous educational resource, guiding learners through neural network construction using TensorFlow. It bridges the gap between deep learning theory and TensorFlow's practical implementation, emphasizing gradient descent, backpropagation, and weight initialization. Bajaj et. al. (2024) [288] implemented CNNs for handwritten digit recognition using TensorFlow and provides a rigorous mathematical breakdown of convolution operations, activation functions, and optimization techniques. It highlights TensorFlow's computational efficiency in large-scale character recognition tasks. Abbass and Fyath (2024) [289] introduced a TensorFlow-based framework for optical fiber communication modeling. It explores how deep learning can optimize fiber optic transmission efficiency by using TensorFlow for predictive analytics and channel equalization. Prabha et. al. (2024) [290] rigorously analyzed TensorFlow's role in precision agriculture, focusing on time-series analysis, computer vision, and reinforcement learning for crop monitoring. It delves into TensorFlow's API optimizations for handling sensor data and remote sensing images. Abdelmadjid and Abdeldjallil (2024) [291] examined TensorFlow Lite for edge computing, rigorously testing optimized CNN architectures on low-power devices. It provides a theoretical comparison of computational efficiency, energy consumption, and model accuracy in resource-constrained environments. Mlambo (2024) [292] bridged Bayesian inference and deep learning, providing a rigorous derivation of Bayesian Neural Networks (BNNs) implemented in TensorFlow. It explores how TensorFlow integrates probabilistic models with deep learning frameworks.

TensorFlow operates primarily on *tensors*, which are multi-dimensional arrays generalizing scalars, vectors, and matrices. For instance, a scalar is a rank-0 tensor, a vector is a rank-1 tensor, a matrix is a rank-2 tensor, and tensors of higher ranks represent multi-dimensional arrays. These tensors can be written mathematically as:

$$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_n} \tag{23.1}$$

where $d_1, d_2, \ldots, d_n$ represent the dimensions of the tensor. TensorFlow leverages efficient *tensor operations* that allow the manipulation of large-scale data in a computationally optimized manner. These operations are the foundation of all the transformations and calculations within TensorFlow models. For example, the *dot product* of two vectors $\vec{a}$ and $\vec{b}$ is a scalar:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i \tag{23.2}$$

Similarly, for matrices, operations like matrix multiplication $A \cdot B$ are highly optimized, taking advantage of *batch processing* and *parallelism* on devices such as GPUs and TPUs. TensorFlow's underlying libraries, such as Eigen, employ these parallel strategies to optimize memory usage and reduce computation time. The heart of TensorFlow's efficiency lies in its *computation graph*, which represents the relationships between different operations. The computation graph is a directed acyclic graph (DAG) where nodes represent computational operations, and the edges represent the flow of data (tensors). Each operation in the graph is a function, f, that maps a set of inputs to an output tensor:

$$y = f(x_1, x_2, \ldots, x_n) \tag{23.3}$$

The graph is built by users or automatically by TensorFlow, where the nodes represent operations such as addition, multiplication, or more complex transformations. Once the computation graph is defined, TensorFlow optimizes the graph by reordering computations, applying algebraic transformations, or parallelizing independent subgraphs. The graph is executed either in a dynamic manner (eager execution) or after optimization (static graph execution), depending on the user's preference. *Automatic differentiation* is another key feature of TensorFlow, and it relies on the *chain rule* of differentiation to compute gradients. The gradient of a scalar-valued function $f(x_1, x_2, \ldots, x_n)$ with respect to an input tensor x_i is computed as:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{23.4}$$

where y_j represents intermediate variables computed during the forward pass of the network. In the context of a neural network, this chain rule is used to propagate errors backward from the output to the input layers during the *backpropagation* process, where the objective is to update the network's weights to minimize the loss function L. Consider a neural network with a simple architecture, consisting of an input layer, one hidden layer, and an output layer. Let X represent the input tensor, W_1 and b_1 the weights and biases of the hidden layer, and W_2 and b_2 the weights and biases of the output layer. The forward pass can be written as:

$$h = \sigma(W_1 X + b_1) \tag{23.5}$$

$$\hat{y} = W_2 h + b_2 \tag{23.6}$$

where σ is the activation function, such as the ReLU function $\sigma(x) = \max(0, x)$, and $\hat{y}$ is the predicted output. The objective in training a model is to minimize a loss function $L(\hat{y}, y)$, where y represents the true labels. The loss function can take different forms, such as the *mean squared error* for regression tasks:

$$L(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{23.7}$$

or the *cross-entropy loss* for classification tasks:

$$L(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{23.8}$$

where C is the number of classes, and $\hat{y}_i$ is the predicted probability of class i under the softmax function. The optimization of this loss function requires the computation of the *gradients* of L with respect to the model parameters W_1, b_1, W_2, b_2. This is achieved through *backpropagation*, which applies the chain rule iteratively through the layers of the network. To perform optimization, TensorFlow employs algorithms like *Gradient Descent (GD)*. The basic gradient descent update rule for parameters θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta) \tag{23.9}$$

where η is the learning rate, and $\nabla_\theta L(\theta)$ represents the gradient of the loss function with respect to the model parameters θ. Variants of gradient descent, such as *Stochastic Gradient Descent (SGD)*, update the parameters using a subset (mini-batch) of the training data rather than the entire dataset:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} L(\theta, x_i, y_i) \tag{23.10}$$

where m is the batch size, and (x_i, y_i) are the data points in the mini-batch. More sophisticated optimizers like *Adam* (Adaptive Moment Estimation) use both momentum (first moment) and scaling (second moment) to adapt the learning rate for each parameter. The update rule for Adam is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta L(\theta) \tag{23.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_\theta L(\theta))^2 \tag{23.12}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{23.13}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{23.14}$$

where β_1 and β_2 are the exponential decay rates, and ϵ is a small constant to prevent division by zero. The inclusion of both the first and second moments allows Adam to adaptively adjust the learning rate, speeding up convergence. In addition to standard optimization methods, TensorFlow supports *distributed computing*, enabling model training across multiple devices, such as GPUs and TPUs. In a distributed setting, the model's parameters are split across different workers, each handling a portion of the data. The gradients computed by each worker are averaged, and the global parameters are updated:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L_i(\theta) \tag{23.15}$$

where $L_i(\theta)$ is the loss computed on the i-th device, and N is the total number of devices. TensorFlow's efficient parallelism ensures that large-scale data processing tasks can be carried out with high computational throughput, thus speeding up model training on large datasets.

TensorFlow also facilitates *model deployment* on different platforms. *TensorFlow Lite* enables model inference on mobile devices by converting trained models into optimized, smaller formats. This process involves *quantization*, which reduces the precision of the weights and activations, thereby reducing memory consumption and computation time. The conversion process aims to balance model accuracy and performance, ensuring that deep learning models can run efficiently on resource-constrained devices like smartphones and IoT devices. For *web applications*, TensorFlow offers *TensorFlow.js*, which allows users to run machine learning models directly in the browser, leveraging the computational power of the client-side GPU or CPU. This is particularly useful for real-time interactions where low-latency predictions are required without sending data to a server. Moreover, TensorFlow provides an ecosystem that extends beyond basic machine learning tasks. For instance, *TensorFlow Extended (TFX)* supports the deployment of machine learning models in production environments, automating the steps from model training to deployment. *TensorFlow Probability* supports probabilistic modeling and uncertainty estimation, which are critical in domains such as reinforcement learning and Bayesian inference.

23.2 PyTorch

Literature Review: Galaxy Yanshi Team of Beihang University [293] examined the use of PyTorch as a deep learning framework for real-time astronaut facial recognition in space stations. It explores the Bayesian coding theory within PyTorch models and its significance in optimizing neural network architectures. It provides a theoretical exploration of probability distributions in PyTorch models, demonstrating how deep learning can be used in constrained computational environments. Tabel (2024) [294] extended PyTorch to Spiking Neural Networks (SNNs), a biologically inspired neural network type. It details a new theoretical approach for learning spike timings using PyTorch's computational graph. The paper bridges neuromorphic computing and PyTorch's automatic differentiation, expanding the theory behind temporal deep learning. Naderi et. al. (2024) [295] introduced a hybrid physics-based deep learning framework that integrates discrete element modeling (DEM) with PyTorch-based networks. It demonstrates how physical simulation problems can be formulated as deep learning models in PyTorch, providing new insights into neural solvers for scientific computing. Polaka (2024) [296] evaluated reinforcement learning (RL) theories within PyTorch, exploring the mathematical rigor of RL frameworks in safe AI applications. The author provided a strong theoretical foundation for understanding deep reinforcement learning (DeepRL) in PyTorch, emphasizing how state-of-the-art RL theories are embedded in the framework. Erdogan et. al. (2024) [297] explored the theoretical framework for reducing stochastic communication overheads in large-scale recommendation systems built using PyTorch. It introduced an optimized gradient synchronization method that can enhance PyTorch-based deep learning models for distributed computing. Liao et. al. (2024) [298] extended the Iterative Partial Diffusion Model (IPDM) framework, implemented in PyTorch, for medical image processing and advanced the theory of deep generative models in PyTorch, specifically in diffusion-based learning techniques. Sekhavat et. al. (2024) [299] examined the theoretical intersection between deep learning in PyTorch and artificial intelligence creativity, referencing Nietzschean philosophical concepts. The author also explored how PyTorch enables neural creativity and provides a rigorous theoretical model for computational aesthetics. Cai et. al. (2025) [300] developed a new theoretical framework for explainability in neural networks using Shapley values, implemented in PyTorch and enhanced the mathematical rigor of explainable AI (XAI) using PyTorch's autograd system to analyze feature importance. Na (2024) [301] proposed a novel ensemble learning theory using PyTorch, specifically in weakly supervised learning (WSL). The paper extends Bayesian learning models in PyTorch for handling sparse labeled data, addressing critical gaps in WSL. Khajah (2024) [302] combined item response theory (IRT) and Bayesian knowledge tracing (BKT) using PyTorch to model generalizable skill discovery. This study presents a rigorous statistical theory for adaptive learning systems using PyTorch's probabilistic programming capabilities.

The **dynamic computation graph** in PyTorch forms the core of its ability to perform efficient and flexible machine learning tasks, especially deep learning models. To understand the underlying mathematical and computational principles, we must explore how the graph operates, what it represents, and how it changes during the execution of a machine learning program. Unlike the static computation graphs employed in frameworks like TensorFlow (pre-Eager execution mode), PyTorch constructs the computation graph dynamically, as the operations are performed in the forward pass. This allows PyTorch to adapt to various input sizes, model structures, and control flows that can change during execution. This adaptability is essential in enabling PyTorch to handle models like recurrent neural networks (RNNs), which operate on sequences of varying lengths, or models that incorporate conditionals in their computation steps.

The **computation graph** itself can be mathematically represented as a **directed acyclic graph (DAG)**, where the nodes represent operations and intermediate results, while the edges represent the flow of data between these nodes. Each operation (e.g., addition, multiplication, or non-linear activation) is applied to tensors, and the outputs of these operations are used as inputs for subsequent operations. The central feature of PyTorch's dynamic computation graph is its **construction at runtime**. For instance, when a tensor $\mathbf{A}$ is created, it might be involved in a series of operations that eventually lead to the calculation of a loss function $\mathcal{L}$. As each operation is executed, PyTorch constructs an edge from the node representing the input tensor $\mathbf{A}$ to the node representing the output tensor $\mathbf{B}$. Mathematically, the transformation between these tensors can be described by:

$$\mathbf{B} = f(\mathbf{A}; \theta) \tag{23.16}$$

where f represents the transformation function (which could be a linear or nonlinear operation), and θ represents the parameters involved in this transformation (e.g., weights or biases in the case of neural networks). The construction of the dynamic graph allows PyTorch to deal with **variable-length sequences**, which are common in tasks such as **time-series prediction**, **natural language processing (NLP)**, and **speech recognition**. The length of the sequence can change depending on the input data, and thus, the number of iterations or layers required in the computation will also vary. In a **recurrent neural network (RNN)**, for example, the hidden state $\mathbf{h}_t$ at each time step t is a function of the previous hidden state $\mathbf{h}_{t-1}$ and the input at the current time step $\mathbf{x}_t$. This can be described mathematically as:

$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \tag{23.17}$$

where f is typically a non-linear activation function (e.g., a hyperbolic tangent or a sigmoid), and $\mathbf{W}_h, \mathbf{W}_x, \mathbf{b}$ represent the weight matrices and bias vector, respectively. This equation encapsulates the recursive nature of RNNs, where each output depends on the previous output and the current input. In a static computation graph, the number of operations for each sequence would need to be predefined, leading to inefficiency when sequences of different lengths are processed. However, in PyTorch, the computation graph is created dynamically for each sequence, which allows for the efficient handling of varying-length sequences and avoids redundant computation.

The key to PyTorch's efficiency lies in **automatic differentiation**, which is managed by its **autograd** system. When a tensor $\mathbf{A}$ has the property `requires_grad=True`, PyTorch starts tracking all operations performed on it. Suppose that the tensor $\mathbf{A}$ is involved in a sequence of operations to compute a scalar loss $\mathcal{L}$. For example, if the loss is a function of $\mathbf{Y}$, the output tensor, which is computed through multiple layers, the objective is to find the gradient of $\mathcal{L}$ with respect to $\mathbf{A}$. This requires the computation of the Jacobian matrix, which represents the gradient of each component of $\mathbf{Y}$ with respect to each component of $\mathbf{A}$. Using the chain rule of differentiation, the gradient of the loss with respect to $\mathbf{A}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \sum_i \frac{\partial \mathcal{L}}{\partial \mathbf{Y}_i} \cdot \frac{\partial \mathbf{Y}_i}{\partial \mathbf{A}} \tag{23.18}$$

This is an application of the **multivariable chain rule**, where $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}_i}$ represents the gradient of the loss with respect to the output tensor at the i-th component, and $\frac{\partial \mathbf{Y}_i}{\partial \mathbf{A}}$ is the Jacobian matrix for the transformation from $\mathbf{A}$ to $\mathbf{Y}$. This computation is achieved by **backpropagating** the gradients through the computation graph that PyTorch builds dynamically. Every operation node in the graph has an associated gradient, which is propagated backward through the graph as we move from the loss back to the input parameters. For example, if $\mathbf{Y} = \mathbf{A} \cdot \mathbf{B}$, the gradient of the loss with respect to $\mathbf{A}$ would be:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \cdot \mathbf{B}^T \tag{23.19}$$

Similarly, the gradient with respect to $\mathbf{B}$ would be:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \cdot \mathbf{A}^T \tag{23.20}$$

This shows how the gradients are passed backward through the computation graph, utilizing the stored operations at each node to calculate the required derivatives. The advantage of this **dynamic construction** of the graph is that it does not require the entire graph to be constructed beforehand, as in the static graph approach. Instead, the graph is dynamically updated as operations are executed, making it both more **memory-efficient** and **computationally efficient**. An important feature of PyTorch's dynamic graph is its ability to handle **conditionals** within the computation. Consider a case where we have different branches in the computation based on a conditional statement. In a static graph, such conditionals would require the entire graph to be predetermined, including all possible branches. In contrast, PyTorch constructs the relevant part of the graph depending on the input data, effectively enabling a **branching computation**. For instance, suppose that we have a decision-making process in a neural network model, where the output depends on whether an input tensor exceeds a threshold $\mathbf{x}_i > \mathbf{t}$:

$$\mathbf{y}_i = \begin{cases} \mathbf{A} \cdot \mathbf{x}_i + \mathbf{b} & \text{if } \mathbf{x}_i > \mathbf{t} \\ \mathbf{C} \cdot \mathbf{x}_i + \mathbf{d} & \text{otherwise} \end{cases} \tag{23.21}$$

In a static graph, we would have to design two separate branches and potentially deal with the computational cost of unused branches. In PyTorch's dynamic graph, only the relevant branch is executed, and the graph is updated accordingly to reflect the necessary operations. The **memory efficiency** in PyTorch's dynamic graph construction is particularly evident when handling large models and training on **large datasets**. When building models like **deep neural networks (DNNs)**, the operations performed on each tensor during both the forward and backward passes are recorded in the computation graph. This allows for **efficient reuse** of intermediate results, and only the necessary memory is allocated for each tensor during the graph's construction. This stands in contrast to static computation graphs, where the full graph needs to be defined and memory allocated up front, potentially leading to unnecessary memory consumption.

To summarize, the **dynamic computation graph** in PyTorch is a powerful tool that allows for flexible model building and efficient computation. By constructing the graph incrementally during the execution of the forward pass, PyTorch is able to dynamically adjust to the input size, control flow, and variable-length sequences, leading to more efficient use of memory and computational resources. The **autograd** system enables **automatic differentiation**, applying the chain rule of calculus to compute gradients with respect to all model parameters. This flexibility is a key reason why PyTorch has gained popularity for deep learning research and production, as it combines **high performance** with **flexibility** and **transparency**, allowing researchers and engineers to experiment with dynamic architectures and complex control flows without sacrificing efficiency.

23.3 JAX

Literature Review: Li et. al. (2024) [313] introduced JAX-based differentiable density functional theory (DFT), enabling end-to-end differentiability in materials science simulations. This paper extends machine learning theory into quantum chemistry by leveraging JAX's automatic differentiation and parallelization capabilities for efficient optimization of density functional models. Bieberich and Li (2024) [314] explored quantum machine learning (QML) using JAX and Diffrax to solve neural differential equations efficiently. They developed a new theoretical model for quantum neural ODEs and discussed how JAX facilitates efficient GPU-based quantum simulations. Dagréou et. al. (2024) [315] analyzed the efficiency of Hessian-vector product (HVP) computation in JAX and PyTorch for deep learning. They established a mathematical foundation for computing second-order derivatives in deep learning and optimization, showcasing JAX's superior automatic differentiation. Lohoff and Neftci (2025) [316] developed a deep reinforcement learning (DRL) model that optimizes JAX's autograd engine for scientific computing. They demonstrated how reinforcement learning improves computational efficiency in JAX through a theoretical framework that eliminates redundant computations in deep learning. Legrand et. al. (2024) [317] introduced a JAX and Rust-based deep learning library for predictive coding networks (PCNs). They explored theoretical extensions of neural networks beyond traditional backpropagation, providing a formalized framework for hierarchical generative models. Alzás and Radev (2024) [318] used JAX to create differentiable models for nuclear reactions, demonstrating its power in high-energy physics simulations. They established a new differentiable framework for theoretical physics, utilizing JAX's gradient-based optimization to improve nuclear physics modeling. Edenhofer et. al. (2024) [319] developed a Gaussian Process and Variational Inference framework in JAX, extending traditional Bayesian methods. They bridged statistical physics and deep learning, formulating a theoretical link between Gaussian processes and deep neural networks using JAX. Chan et. al. (2024) [320] proposed a JAX-based quantum machine learning framework for long-tailed X-ray classification. They introduced a novel quantum transfer learning technique within JAX, demonstrating its advantages over classical deep learning models in medical imaging. Ye et. al. (2025) [321] used JAX to model electron transfer kinetics, bridging deep learning and density functional theory (DFT). They developed a new theoretical framework for modeling charge transfer reactions, leveraging JAX's high-performance computation for quantum chemistry applications. Khan et. al. (2024) [322] extended NODEs using JAX's efficient autodiff capabilities for high-dimensional dynamical systems. They established a rigorous mathematical framework for extending NODEs to stochastic and chaotic systems, leveraging JAX's high-speed parallelization.

JAX is an advanced numerical computing framework designed to optimize high-performance scientific computing tasks with particular emphasis on automatic differentiation, hardware acceleration, and just-in-time (JIT) compilation. These capabilities are essential for applications in machine learning, optimization, physical simulations, and computational science, where large-scale, high-dimensional computations must be executed with both speed and efficiency. At its core, JAX integrates a deep mathematical structure based on advanced concepts in linear algebra, optimization theory, tensor calculus, and numerical differentiation, providing the foundation for scalable computations across multi-core CPUs, GPUs, and TPUs. The framework leverages the power of reverse-mode differentiation and JIT compilation to significantly reduce computation time while ensuring correctness and accuracy. The following rigorous exploration will dissect these operations mathematically and conceptually, explaining their inner workings and theoretical implications.

JAX's **automatic differentiation** is central to its ability to compute gradients, Jacobians, Hessians, and other derivatives efficiently. For many applications, the function of interest involves computing gradients with respect to model parameters in optimization and machine learning tasks. Automatic differentiation allows for the efficient computation of these gradients using the **reverse-mode** differentiation technique. Let us consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$, and suppose we wish to compute the gradient of the scalar-valued output with respect to each input variable. The gradient of f, denoted as $\nabla_{\mathbf{x}} f$, is a vector of partial derivatives:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left(\frac{\partial f_1}{\partial x_1}, \frac{\partial f_1}{\partial x_2}, \dots, \frac{\partial f_1}{\partial x_n}, \dots, \frac{\partial f_m}{\partial x_n} \right), \tag{23.22}$$

where $f = (f_1, f_2, \dots, f_m)$ represents a vector of m scalar outputs, and $\mathbf{x} = (x_1, x_2, \dots, x_n)$ represents the input vector. Reverse-mode differentiation computes this gradient by applying the **chain rule** in reverse order. If f is composed of several intermediate functions, say $f = g \circ h$, where $g : \mathbb{R}^m \to \mathbb{R}^p$ and $h : \mathbb{R}^n \to \mathbb{R}^m$, the gradient of f with respect to $\mathbf{x}$ is computed recursively by applying the chain rule:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left(\frac{\partial g}{\partial h} \right) \cdot \left(\frac{\partial h}{\partial x_1}, \frac{\partial h}{\partial x_2}, \dots, \frac{\partial h}{\partial x_n} \right). \tag{23.23}$$

This recursive application of the chain rule ensures that each intermediate gradient computation is propagated backward through the function's layers, reducing the number of required passes compared to forward-mode differentiation. This technique becomes particularly beneficial for functions where the number of outputs m is much smaller than the number of inputs n, as it minimizes the computational complexity. In the context of JAX, automatic differentiation is utilized through functions like `jax.grad`, which can be applied to scalar-valued functions to return their gradients with respect to vector-valued inputs. To compute higher-order derivatives, such as the Hessian matrix, JAX allows for the computation of second- and higher-order derivatives using similar principles. The Hessian matrix H of a scalar function $f(\mathbf{x})$ is given by the matrix of second derivatives:

$$H = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right), \tag{23.24}$$

which is computed by applying the chain rule once again. The second-order derivatives can be computed efficiently by differentiating the gradient once more, and this process can be extended to higher-order derivatives by continuing the recursive application of the chain rule. A central concept in JAX's approach to high-performance computing is **JIT (just-in-time) compilation**, which provides substantial performance gains by compiling Python functions into optimized machine code tailored to the underlying hardware architecture. JIT compilation in JAX is built on the foundation of the **XLA (Accelerated Linear Algebra)** compiler. XLA optimizes the execution of tensor operations by fusing multiple operations into a single kernel, thereby reducing the overhead associated with launching individual computation kernels. This technique is particularly effective for matrix multiplications, convolutions, and other tensor operations commonly found in machine learning tasks. For example, consider a simple sequence of operations $f = \text{Op}_1(\text{Op}_2(\dots(\text{Op}_n(\mathbf{x}))))$, where Op_i represents different mathematical operations applied to the input tensor $\mathbf{x}$. Without optimization, each operation would typically be executed separately, introducing significant overhead. JAX's JIT compiler, however, recognizes this sequence and applies a fusion transformation, resulting in a single composite operation:

$$\text{Optimized}(f(\mathbf{x})) = \text{Fused Op}(\mathbf{x}), \tag{23.25}$$

where Fused Op represents a highly optimized version of the original sequence of operations. This optimization minimizes the number of kernel launches and reduces memory access overhead, which in turn accelerates the computation. The JIT compiler analyzes the computational graph of the function and identifies opportunities to combine operations into a more efficient form, ultimately speeding up the computation on hardware accelerators such as GPUs or TPUs.

The **vectorization** capability provided by JAX through the `jax.vmap` operator is another essential optimization for high-performance computing. This feature automatically vectorizes functions across batches of data, allowing the same operation to be applied in parallel across multiple data points. Mathematically, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a batch of inputs $\mathbf{X} \in \mathbb{R}^{B \times n}$, the vectorized function can be expressed as:

$$\mathbf{Y} = \text{vmap}(f)(\mathbf{X}), \tag{23.26}$$

where B is the batch size and $\mathbf{Y}$ is the matrix in $\mathbb{R}^{B \times m}$, containing the results of applying f to each row of $\mathbf{X}$. The mathematical operation applied by JAX is the same as applying f to each individual row $\mathbf{X}_i$, but with the benefit that the entire batch is processed in parallel, exploiting the available hardware resources efficiently. The ability to **parallelize computations across multiple devices** is one of JAX's strongest features, and it is enabled through the `jax.pmap` operator. This operator allows for the parallel execution of functions across different devices, such as multiple GPUs or TPUs. Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a batch of inputs $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_p)$, distributed across p devices. The parallelized execution of the function can be written as:

$$\mathbf{Y} = \text{pmap}(f)(\mathbf{X}), \tag{23.27}$$

where each device independently computes its portion of the computation $f(\mathbf{X}_i)$, and the results are gathered into the final output $\mathbf{Y}$. This capability is essential for large-scale distributed training of machine learning models, where the model's parameters and data must be distributed across multiple devices to ensure efficient training. The parallelization effectively reduces computation time, as each device operates on a distinct subset of the data and model parameters. **GPU/TPU acceleration** is another crucial aspect of JAX's performance, and it is facilitated by libraries like cuBLAS for GPUs, which are specifically designed to optimize matrix operations. The primary operation used in many numerical computing tasks is matrix multiplication, and JAX optimizes this by leveraging hardware-accelerated implementations of these operations. Consider the matrix multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $\mathbf{B} \in \mathbb{R}^{m\times p}$, resulting in a matrix $\mathbf{C} \in \mathbb{R}^{n\times p}$:

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}. \tag{23.28}$$

Using cuBLAS or a similar library, JAX can execute this operation on a GPU, utilizing the massive parallel processing power of the hardware to perform the multiplication efficiently. This operation can be further optimized by considering the specific memory hierarchies of GPUs, where large matrix multiplications are broken down into smaller tiles that fit into the GPU's high-speed memory. This technique minimizes memory bandwidth constraints, accelerating the computation. In addition to these core operations, JAX allows for the definition of **custom gradients** using the `jax.custom_jvp` decorator, which enables users to specify the Jacobian-vector products (JVPs) manually for more efficient gradient computation. This feature is especially useful in machine learning applications, where certain operations might have custom gradients that cannot be computed automatically. For instance, in a non-trivial activation function such as the softmax, the custom gradient function might be provided explicitly for efficiency:

$$\frac{\partial \text{softmax}(\mathbf{x})}{\partial \mathbf{x}} = \text{diag}(\text{softmax}(\mathbf{x})) - \text{softmax}(\mathbf{x}) \cdot \text{softmax}(\mathbf{x})^T. \tag{23.29}$$

Thus, JAX allows for both flexibility and performance, enabling scientific computing applications that require both efficiency and the ability to define complex, custom derivatives.

By providing advanced capabilities such as automatic differentiation, JIT compilation, vectorization, parallelization, hardware acceleration, and custom gradients, JAX is equipped to handle a wide range of high-performance computing tasks, making it an invaluable tool for solving complex scientific and engineering problems. The framework not only ensures the correctness of numerical methods but also leverages the power of modern hardware to achieve performance that is crucial for large-scale simulations, machine learning, and optimization tasks.

24 Appendix

24.1 Linear Algebra Essentials

24.1.1 Matrices and Vector Spaces

Definition of a Matrix: A **matrix** A is a rectangular array of numbers (or elements from a field $\mathbb{F}$), arranged in rows and columns:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{F}^{m \times n} \tag{24.1}$$

where a_{ij} denotes the **entry** of A at the i-th row and j-th column. A **square matrix** is one where $m = n$. A matrix is **diagonal** if all off-diagonal entries are zero. For matrices $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{m\times n}$ the following are the matrix operations:

- **Addition:** Defined entrywise:

$$(A + B)_{ij} = A_{ij} + B_{ij} \tag{24.2}$$

- **Scalar Multiplication:** For $\alpha \in \mathbb{F}$,

$$(\alpha A)_{ij} = \alpha \cdot A_{ij} \tag{24.3}$$

- **Matrix Multiplication:** If $A \in \mathbb{F}^{m\times p}$ and $B \in \mathbb{F}^{p\times n}$, then the product $C = AB$ is given by:

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj} \tag{24.4}$$

  This is **only defined when** the number of columns of A equals the number of rows of B.

- **Transpose:** The **transpose** of A, denoted A^T, satisfies:

$$(A^T)_{ij} = A_{ji} \tag{24.5}$$

- **Determinant:** If $A \in \mathbb{F}^{n\times n}$, then its **determinant** is given recursively by:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) \tag{24.6}$$

  where A_{1j} is the $(n-1) \times (n-1)$ submatrix obtained by removing the first row and j-th column.

- **Inverse:** A square matrix A is **invertible** if there exists A^{-1} such that:

$$AA^{-1} = A^{-1}A = I \tag{24.7}$$

  where I is the identity matrix.

24.1.2 Vector Spaces and Linear Transformations

Vector Spaces A **vector space** over a field $\mathbb{F}$ is a set V with two operations:

- **Vector Addition:** $\mathbf{v} + \mathbf{w}$ for $\mathbf{v}, \mathbf{w} \in V$
- **Scalar Multiplication:** $\alpha\mathbf{v}$ for $\alpha \in \mathbb{F}$ and $\mathbf{v} \in V$

satisfying the **8 vector space axioms** (associativity, commutativity, existence of identity, etc.). A set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is a **basis** if:

- It is **linearly independent**:

$$\sum_{i=1}^{n} \alpha_i \mathbf{v}_i = 0 \Rightarrow \alpha_i = 0, \forall i \tag{24.8}$$

- It **spans** V, meaning every $\mathbf{v} \in V$ can be written as:

$$\mathbf{v} = \sum_{i=1}^{n} \beta_i \mathbf{v}_i \tag{24.9}$$

The **dimension** of V, denoted $\dim(V)$, is the number of basis vectors. **Linear Transformations:** A function $T : V \to W$ is **linear** if:

$$T(\alpha\mathbf{v} + \beta\mathbf{w}) = \alpha T(\mathbf{v}) + \beta T(\mathbf{w}) \tag{24.10}$$

The **matrix representation** of T is the matrix A such that:

$$T(\mathbf{x}) = A\mathbf{x} \tag{24.11}$$

24.1.3 Eigenvalues and Eigenvectors

Definition: For a square matrix $A \in \mathbb{F}^{n\times n}$, an **eigenvalue** λ and **eigenvector** $\mathbf{v} \neq \mathbf{0}$ satisfy:

$$A\mathbf{v} = \lambda\mathbf{v} \tag{24.12}$$

Characteristic Equation: The eigenvalues are found by solving:

$$\det(A - \lambda I) = 0 \tag{24.13}$$

which gives an n**-th degree polynomial** in λ. The set of all solutions $\mathbf{v}$ to $(A - \lambda I)\mathbf{v} = 0$ is the **eigenspace** associated with λ.

24.1.4 Singular Value Decomposition (SVD)

Definition: For any $A \in \mathbb{F}^{m\times n}$, the **Singular Value Decomposition** (SVD) states:

$$A = U\Sigma V^T \tag{24.14}$$

where $U \in \mathbb{F}^{m\times m}$ is **orthogonal** ($U^T U = I$), $V \in \mathbb{F}^{n\times n}$ is **orthogonal** ($V^T V = I$), Σ is an $m \times n$ **diagonal matrix**:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \tag{24.15}$$

where σ_i are the **singular values**, given by:

$$\sigma_i = \sqrt{\lambda_i} \tag{24.16}$$

where λ_i are the eigenvalues of $A^T A$.

24.2 Probability and Statistics

24.2.1 Probability Distributions

A **probability distribution** is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment. A random variable X can take values from a sample space S, and the probability distribution describes how the probabilities are distributed over these possible outcomes.

Discrete Probability Distributions: For a discrete random variable X, which takes values from a countable set, the **probability mass function** (PMF) is defined as:

$$P(X = x_i) = p(x_i), \quad \forall x_i \in S \tag{24.17}$$

The PMF satisfies the following properties:

- $0 \leq p(x_i) \leq 1$ for each $x_i \in S$.
- The sum of probabilities across all possible outcomes is 1:

$$\sum_{x_i \in S} p(x_i) = 1 \tag{24.18}$$

An example of a discrete probability distribution is the **binomial distribution**, which describes the number of successes in a fixed number of independent Bernoulli trials. The PMF for the binomial distribution is:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \ldots, n \tag{24.19}$$

Where n is the number of trials, p is the probability of success on each trial, and k is the number of successes.

Continuous Probability Distributions: For a continuous random variable X, which takes values from a continuous set (e.g., the real line), the **probability density function** (PDF) is used instead of the PMF. The PDF $f(x)$ is defined such that for any interval $[a, b]$, the probability that X lies in this interval is:

$$P(a \leq X \leq b) = \int_a^b f(x)\, dx \tag{24.20}$$

The PDF must satisfy:

- $f(x) \geq 0$ for all x.
- The total probability over the entire range of X is 1:

$$\int_{-\infty}^{\infty} f(x)\, dx = 1 \tag{24.21}$$

An example of a continuous probability distribution is the **normal distribution**, which is given by the PDF:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R} \tag{24.22}$$

Where μ is the mean and σ^2 is the variance of the distribution.

24.2.2 Bayes' Theorem

Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. It is a fundamental result in the field of probability theory and statistics.

Let A and B be two events. Then, Bayes' theorem gives the conditional probability of A given B:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{24.23}$$

where $P(A|B)$ is the posterior probability of A given B, $P(B|A)$ is the likelihood, the probability of observing B given A, $P(A)$ is the prior probability of A, $P(B)$ is the marginal likelihood of B, computed as:

$$P(B) = \sum_i P(B|A_i)P(A_i) \tag{24.24}$$

In the continuous case, Bayes' theorem is written as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\int P(B|A')P(A')\,dA'} \tag{24.25}$$

This allows one to update beliefs about a hypothesis A based on observed evidence B. Let us consider a diagnostic test for a disease. Let A be the event that a person has the disease and B be the event that the test is positive. We are interested in the probability that a person has the disease given that the test is positive, i.e., $P(A|B)$. By Bayes' theorem, we have:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{24.26}$$

where $P(B|A)$ is the probability of a positive test result given that the person has the disease (sensitivity), $P(A)$ is the prior probability of having the disease, $P(B)$ is the total probability of a positive test result.

24.2.3 Statistical Measures

Statistical measures summarize the properties of data or a probability distribution. Some key statistical measures are the **mean**, **variance**, **standard deviation**, and **skewness**.

A **statistical measure** is a function $M : \mathcal{S} \to \mathbb{R}$ that assigns a real-valued quantity to an element in a statistical space $\mathcal{S}$, where $\mathcal{S}$ can represent a dataset, a probability distribution, or a stochastic process. Mathematically, a statistical measure must satisfy certain properties such as **measurability**, **invariance under transformation**, and **convergence consistency** in order to be well-defined. Statistical measures can be broadly classified into:

1. **Measures of Central Tendency** (e.g., mean, median, mode)

2. **Measures of Dispersion** (e.g., variance, standard deviation, interquartile range)

3. **Measures of Shape** (e.g., skewness, kurtosis)

4. **Measures of Association** (e.g., covariance, correlation)

5. **Information-Theoretic Measures** (e.g., entropy, mutual information)

Each of these measures provides different insights into the characteristics of a dataset or a probability distribution. There are several Measures of Central Tendency. Given a probability space $(\Omega, \mathcal{F}, P)$ and a random variable $X : \Omega \to \mathbb{R}$, the **expectation** (or mean) is defined as:

$$\mathbb{E}[X] = \int_\Omega X(\omega) dP(\omega) \tag{24.27}$$

If X is a discrete random variable with probability mass function $p(x)$, then:

$$\mathbb{E}[X] = \sum_{x \in \mathbb{R}} x p(x) \tag{24.28}$$

If X is a continuous random variable with probability density function $f(x)$, then:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) dx \tag{24.29}$$

The **median** m of a probability distribution is defined as:

$$P(X \leq m) \geq \frac{1}{2}, \quad P(X \geq m) \geq \frac{1}{2} \tag{24.30}$$

In terms of the cumulative distribution function $F(x)$, the median m satisfies:

$$F(m) = \frac{1}{2} \tag{24.31}$$

The **mode** is defined as the point x_m that maximizes the probability density function:

$$x_m = \arg\max_x f(x) \tag{24.32}$$

The variance σ^2 of a random variable X is given by:

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{24.33}$$

Expanding this expression:

$$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{24.34}$$

The **standard deviation** σ is defined as the square root of the variance:

$$\sigma = \sqrt{\text{Var}(X)} \tag{24.35}$$

If Q_1 and Q_3 denote the first and third quartiles of a dataset (where Q_1 is the 25th percentile and Q_3 is the 75th percentile), then the interquartile range is:

$$IQR = Q_3 - Q_1 \tag{24.36}$$

The **skewness** of a random variable X is defined as:

$$\gamma_1 = \frac{\mathbb{E}[(X - \mathbb{E}[X])^3]}{\sigma^3} \tag{24.37}$$

It quantifies the asymmetry of the probability distribution. The **kurtosis** is given by:

$$\gamma_2 = \frac{\mathbb{E}[(X - \mathbb{E}[X])^4]}{\sigma^4} \tag{24.38}$$

A normal distribution has $\gamma_2 = 3$, and deviations from this indicate whether a distribution has heavy or light tails. There are several Measures of Association. The Covariance is defined as follows: Given two random variables X and Y, their **covariance** is:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{24.39}$$

Expanding:

$$\mathrm{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{24.40}$$

The Pearson Correlation Coefficient defined as:

$$\rho(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{24.41}$$

where σ_X and σ_Y are the standard deviations of X and Y, respectively. The Information-Theoretic Measure is Entropy which is defined as the **entropy** of a discrete probability distribution $p(x)$ is given by:

$$H(X) = -\sum_x p(x) \log p(x) \tag{24.42}$$

For continuous distributions with density $f(x)$, the **differential entropy** is:

$$h(X) = -\int_{-\infty}^{\infty} f(x) \log f(x)\, dx \tag{24.43}$$

Given two random variables X and Y, their mutual information is:

$$I(X; Y) = H(X) + H(Y) - H(X, Y) \tag{24.44}$$

which measures how much knowing X reduces uncertainty about Y. Statistical Measures satisfy Linearity and Invariance i.e.

- **Expectation is linear:**
$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{24.45}$$

- **Variance is translation invariant but scales quadratically:**
$$\mathrm{Var}(aX + b) = a^2 \mathrm{Var}(X) \tag{24.46}$$

For the Convergence and Asymptotic Behavior, The **law of large numbers** ensures that empirical means converge to the expected value, while the **central limit theorem** states that sums of i.i.d. random variables converge in distribution to a normal distribution.

The **mean** or **expected value** of a random variable X, denoted by $\mathbb{E}[X]$, represents the average value of X. For a discrete random variable:

$$\mathbb{E}[X] = \sum_{x_i \in S} x_i p(x_i) \tag{24.47}$$

For a continuous random variable, the expected value is given by:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x)\, dx \tag{24.48}$$

The **variance** of a random variable X, denoted by $\mathrm{Var}(X)$, measures the spread or dispersion of the distribution. For a discrete random variable:

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{24.49}$$

For a continuous random variable:

$$\mathrm{Var}(X) = \int_{-\infty}^{\infty} x^2 f(x)\, dx - \left(\int_{-\infty}^{\infty} x f(x)\, dx \right)^2 \tag{24.50}$$

The **standard deviation** is the square root of the variance and provides a measure of the spread of the distribution in the same units as the random variable:

$$\mathrm{SD}(X) = \sqrt{\mathrm{Var}(X)} \tag{24.51}$$

The **skewness** of a random variable X quantifies the asymmetry of the probability distribution. It is defined as:

$$\mathrm{Skew}(X) = \frac{\mathbb{E}[(X - \mathbb{E}[X])^3]}{(\mathrm{Var}(X))^{3/2}} \tag{24.52}$$

A positive skew indicates that the distribution has a long tail on the right, while a negative skew indicates a long tail on the left. The **kurtosis** of a random variable X measures the "tailedness" of the distribution, i.e., how much of the probability mass is concentrated in the tails. It is defined as:

$$\mathrm{Kurt}(X) = \frac{\mathbb{E}[(X - \mathbb{E}[X])^4]}{(\mathrm{Var}(X))^2} \tag{24.53}$$

A distribution with high kurtosis has heavy tails, and one with low kurtosis has light tails compared to a normal distribution.

24.3 Optimization Techniques

24.3.1 Gradient Descent (GD)

Gradient Descent is an iterative optimization algorithm used to minimize a differentiable function. The goal is to find the point where the function achieves its minimum value. Mathematically, it can be formulated as follows. Given a differentiable objective function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient descent update rule is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k) \tag{24.54}$$

where:

- $\mathbf{x}_k \in \mathbb{R}^n$ is the current point in the n-dimensional space (iteration index k),
- $\nabla f(\mathbf{x}_k)$ is the gradient of the objective function at $\mathbf{x}_k$,
- η is the learning rate (step size).

To analyze the convergence of gradient descent, we assume f is **convex** and **differentiable** with a **Lipschitz continuous gradient**. That is, there exists a constant $L > 0$ such that:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \tag{24.55}$$

This property ensures the gradient of f does not change too rapidly, which allows us to bound the convergence rate. The following is an upper bound on the decrease in the function value at each iteration:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) \leq (1 - \eta L)(f(\mathbf{x}_k) - f(\mathbf{x}^*)), \tag{24.56}$$

where $\mathbf{x}^*$ is the global minimum. Thus, we have the following convergence rate:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq (1 - \eta L)^k (f(\mathbf{x}_0) - f(\mathbf{x}^*)). \tag{24.57}$$

For this to converge, we require $\eta L < 1$. Hence, the step size η must be chosen carefully to ensure convergence.

24.3.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a variant of gradient descent that approximates the gradient of the objective function using a randomly chosen subset (mini-batch) of the data at each iteration. This can significantly reduce the computational cost when the dataset is large.

Let the objective function be the sum of individual functions $f_i(\mathbf{x})$ corresponding to each data point:

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x}), \tag{24.58}$$

where m is the number of data points. In **Stochastic Gradient Descent**, the update rule becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f_{i_k}(\mathbf{x}_k), \tag{24.59}$$

where i_k is a randomly chosen index at the k-th iteration, and $\nabla f_{i_k}(\mathbf{x})$ is the gradient of the function $f_{i_k}(\mathbf{x})$ corresponding to that randomly selected data point. The stochastic gradient is given by:

$$\nabla f_{i_k}(\mathbf{x}_k) = \nabla f_i(\mathbf{x}_k). \tag{24.60}$$

Given that the gradient is stochastic, the convergence analysis of SGD is more complex. Assuming that each f_i is convex and differentiable, and using the strong convexity assumption (i.e., there exists a constant $m > 0$ such that f satisfies the inequality):

$$f(\mathbf{x}) - f(\mathbf{y}) \geq m\|\mathbf{x} - \mathbf{y}\|^2, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \tag{24.61}$$

SGD converges to the optimal solution at a rate of:

$$\mathbb{E}[f(\mathbf{x}_k) - f(\mathbf{x}^*)] \leq \frac{C}{k}, \tag{24.62}$$

where C is a constant depending on the step size η, the variance of the stochastic gradients, and the strong convexity constant m. This slower convergence rate is due to the inherent noise in the gradient estimates. Variance reduction techniques such as **mini-batch SGD** (using multiple data points per iteration) or **Momentum** (accumulating past gradients) are often employed to improve convergence speed and stability.

24.3.3 Second-Order Methods

Second-order methods make use of not just the gradient $\nabla f(\mathbf{x})$, but also the **Hessian matrix** $\mathbf{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$, which is the matrix of second-order partial derivatives of the objective function. The update rule for second-order methods is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k), \tag{24.63}$$

where $\mathbf{H}^{-1}(\mathbf{x}_k)$ is the inverse of the Hessian matrix.

Second-order methods typically have faster convergence rates compared to gradient descent, particularly when the function f has well-conditioned curvature. However, computing the Hessian is computationally expensive, which limits the scalability of these methods. Newton's method is a widely used second-order optimization technique that uses both the gradient and the Hessian. The update rule is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k). \tag{24.64}$$

Newton's method converges quadratically near the optimal point under the assumption that the objective function is twice continuously differentiable and the Hessian is positive definite. More formally, if $\mathbf{x}_k$ is sufficiently close to the optimal point $\mathbf{x}^*$, then the error $\|\mathbf{x}_k - \mathbf{x}^*\|$ decreases quadratically:

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq C\|\mathbf{x}_k - \mathbf{x}^*\|^2, \tag{24.65}$$

where C is a constant depending on the condition number of the Hessian.

Since directly computing the Hessian is expensive, quasi-Newton methods aim to approximate the inverse Hessian at each iteration. One of the most popular quasi-Newton methods is the **Broyden–Fletcher–Goldfarb–Shanno (BFGS)** method, which maintains an approximation to the inverse Hessian, updating it at each iteration. The Summary of what we discussed above are as follows:

- **Gradient Descent (GD):** An optimization algorithm that updates the parameter vector in the direction opposite to the gradient of the objective function. Convergence is guaranteed under convexity assumptions with an appropriately chosen step size.
- **Stochastic Gradient Descent (SGD):** A variant of GD that uses a random subset of the data to estimate the gradient at each iteration. While faster and less computationally intensive, its convergence is slower and more noisy, requiring variance reduction techniques for efficient training.
- **Second-Order Methods:** These methods use the Hessian (second derivatives of the objective function) to accelerate convergence, often exhibiting quadratic convergence near the optimum. However, the computational cost of calculating the Hessian restricts their practical use. Quasi-Newton methods, such as BFGS, approximate the Hessian to improve efficiency.

Each of these methods has its advantages and trade-offs, with gradient-based methods being widely used due to their simplicity and efficiency, and second-order methods providing faster convergence but at higher computational costs.

24.4 Matrix Calculus

24.4.1 Matrix Differentiation

Consider a matrix $\mathbf{A}$ of size $m \times n$, where $A = [a_{ij}]$. For the purposes of differentiation, we will focus on functions $f(\mathbf{A})$ that map matrices to scalars or other matrices. We aim to compute the derivative of $f(\mathbf{A})$ with respect to $\mathbf{A}$. Let $f(\mathbf{A})$ be a scalar function of the matrix $\mathbf{A}$. The derivative of this scalar function with respect to $\mathbf{A}$ is defined as:

$$\frac{\partial f(\mathbf{A})}{\partial \mathbf{A}} = \left[\frac{\partial f(\mathbf{A})}{\partial a_{ij}}\right] \tag{24.66}$$

This is a matrix where the (i, j)-th entry is the partial derivative of the scalar function with respect to the element a_{ij}. Let us take an example of Differentiating the Frobenius Norm. Consider the Frobenius norm of a matrix $\mathbf{A}$, defined as:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} \tag{24.67}$$

To compute the derivative of $\|\mathbf{A}\|_F$ with respect to $\mathbf{A}$, we first apply the chain rule:

$$\frac{\partial \|\mathbf{A}\|_F}{\partial a_{ij}} = \frac{2a_{ij}}{2\|\mathbf{A}\|_F} = \frac{a_{ij}}{\|\mathbf{A}\|_F} \tag{24.68}$$

Thus, the gradient of the Frobenius norm is the matrix $\frac{\mathbf{A}}{\|\mathbf{A}\|_F}$. The Matrix Derivatives of Common Functions are as follows:

- **Matrix trace**: For a matrix $\mathbf{A}$, the derivative of the trace $\mathrm{Tr}(\mathbf{A})$ with respect to $\mathbf{A}$ is the identity matrix:
$$\frac{\partial \mathrm{Tr}(\mathbf{A})}{\partial \mathbf{A}} = \mathbf{I} \tag{24.69}$$
- **Matrix product**: Let $\mathbf{A}$ and $\mathbf{B}$ be matrices, and consider the product $f(\mathbf{A}) = \mathbf{AB}$. The derivative of this product with respect to $\mathbf{A}$ is:
$$\frac{\partial (\mathbf{AB})}{\partial \mathbf{A}} = \mathbf{B}^T \tag{24.70}$$
- **Matrix inverse**: The derivative of the inverse of $\mathbf{A}$ with respect to $\mathbf{A}$ is:
$$\frac{\partial (\mathbf{A}^{-1})}{\partial \mathbf{A}} = -\mathbf{A}^{-1}\left(\frac{\partial \mathbf{A}}{\partial \mathbf{A}}\right)\mathbf{A}^{-1} \tag{24.71}$$

24.4.2 Tensor Differentiation

A **tensor** is a multi-dimensional array of components that transform according to certain rules under a change of basis. For simplicity, let's focus on second-order tensors (which are matrices in $m \times n$ form), but the results can extend to higher-order tensors.

Let $\mathbf{T}$ be a tensor, represented by the array of components $T_{i_1,i_2,\ldots,i_k}$, where the indices $i_1, i_2, \ldots, i_k$ are the dimensions of the tensor. Let $f(\mathbf{T})$ be a scalar-valued function that depends on the tensor $\mathbf{T}$. The derivative of this function with respect to the tensor components $T_{i_1,\ldots,i_k}$ is given by:
$$\frac{\partial f(\mathbf{T})}{\partial T_{i_1,\ldots,i_k}} = \text{Jacobian of}\, f(\mathbf{T})\,\text{with respect to}\, T_{i_1,\ldots,i_k} \tag{24.72}$$

For example, consider a function of a second-order tensor, $f(\mathbf{T})$, where $\mathbf{T}$ is a matrix. The differentiation rule follows similar principles as matrix differentiation. The Jacobian is computed for each tensor component in the same fashion, based on the partial derivatives with respect to the individual tensor components.

Consider a second-order tensor $\mathbf{T}$, and let's compute the derivative of the Frobenius norm of $\mathbf{T}$:
$$\|\mathbf{T}\|_F = \sqrt{\sum_{i_1,i_2,\ldots,i_k} T^2_{i_1,\ldots,i_k}} \tag{24.73}$$
Differentiating with respect to $T_{i_1,\ldots,i_k}$, we get:
$$\frac{\partial \|\mathbf{T}\|_F}{\partial T_{i_1,\ldots,i_k}} = \frac{2T_{i_1,\ldots,i_k}}{2\|\mathbf{T}\|_F} = \frac{T_{i_1,\ldots,i_k}}{\|\mathbf{T}\|_F} \tag{24.74}$$
This is the gradient of the Frobenius norm, where each component $T_{i_1,\ldots,i_k}$ is normalized by the Frobenius norm. For higher-order tensors, differentiation follows the same principles but extends to multi-indexed components. If $\mathbf{T}$ is a third-order tensor, say T_{i_1,i_2,i_3}, the differentiation of $f(\mathbf{T})$ with respect to any component is given by:
$$\frac{\partial f(\mathbf{T})}{\partial T_{i_1,i_2,i_3}} = \text{Jacobian of}\, f(\mathbf{T})\,\text{with respect to the multi-index components.} \tag{24.75}$$
For the tensor product of two tensors $\mathbf{T}_1$ and $\mathbf{T}_2$, say of orders p and q, respectively, the product is another tensor of order $p + q$. Differentiation of the tensor product $\mathbf{T}_1 \otimes \mathbf{T}_2$ follows the product rule:
$$\frac{\partial (\mathbf{T}_1 \otimes \mathbf{T}_2)}{\partial \mathbf{T}_1} = \mathbf{T}_2, \qquad \frac{\partial (\mathbf{T}_1 \otimes \mathbf{T}_2)}{\partial \mathbf{T}_2} = \mathbf{T}_1 \tag{24.76}$$

This tensor product rule applies for higher-order tensors, where differentiation follows tensor contraction rules. The process of differentiating matrices and tensors extends the rules of differentiation to multi-dimensional data structures, with careful application of chain rules, product rules, and understanding the Jacobian of the functions. For matrices, the derivative is a matrix of partial derivatives, while for tensors, the derivative is typically expressed as a tensor with respect to multi-index components. In higher-order tensor differentiation, we apply these principles recursively, accounting for multi-index notation, and respecting the tensor contraction rules that define how the components interact.

We start with the Differentiation of Scalar-Valued Functions with Matrix Arguments. Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ be a scalar function of a matrix $\mathbf{X}$. The **differential** df of f is defined by:

$$df = \lim_{\|\mathbf{H}\|\to 0} \frac{f(\mathbf{X}+\mathbf{H}) - f(\mathbf{X})}{\|\mathbf{H}\|} \tag{24.77}$$

where $\mathbf{H}$ is an infinitesimal perturbation. The total derivative of f is given by:

$$df = \operatorname{tr}\left(\left(\frac{\partial f}{\partial \mathbf{X}}\right)^T d\mathbf{X}\right). \tag{24.78}$$

Definition of the Matrix Gradient: The **gradient $\mathbf{D}_{\mathbf{X}}f$** (or **Jacobian**) is the unique matrix satisfying:

$$df = \operatorname{tr}\left(\mathbf{D}_{\mathbf{X}}^T d\mathbf{X}\right). \tag{24.79}$$

This ensures that differentiation is **dual** to the Frobenius inner product $\langle \mathbf{A}, \mathbf{B}\rangle = \operatorname{tr}(\mathbf{A}^T\mathbf{B})$, giving a **Hilbert space structure**. Let's start with the example of Quadratic Form Differentiation. Let $f(\mathbf{X}) = \operatorname{tr}(\mathbf{X}^T\mathbf{A}\mathbf{X})$. Expanding in a small perturbation $\mathbf{H}$:

$$f(\mathbf{X}+\mathbf{H}) = \operatorname{tr}((\mathbf{X}+\mathbf{H})^T\mathbf{A}(\mathbf{X}+\mathbf{H})). \tag{24.80}$$

Expanding and isolating linear terms:

$$df = \operatorname{tr}(\mathbf{H}^T\mathbf{A}\mathbf{X}) + \operatorname{tr}(\mathbf{X}^T\mathbf{A}\mathbf{H}). \tag{24.81}$$

Using the cyclic property of the trace:

$$df = \operatorname{tr}(\mathbf{H}^T(\mathbf{A}\mathbf{X} + \mathbf{A}^T\mathbf{X})). \tag{24.82}$$

Thus, the derivative is:

$$\frac{\partial f}{\partial \mathbf{X}} = \mathbf{A}\mathbf{X} + \mathbf{A}^T\mathbf{X}. \tag{24.83}$$

If $\mathbf{A}$ is symmetric ($\mathbf{A}^T = \mathbf{A}$), this simplifies to:

$$\frac{\partial f}{\partial \mathbf{X}} = 2\mathbf{A}\mathbf{X}. \tag{24.84}$$

Regarding the Differentiation of Matrix-Valued Functions. Consider a differentiable function $\mathbf{F} : \mathbb{R}^{m\times n} \to \mathbb{R}^{p\times q}$. The **Fréchet derivative $\mathcal{D}_{\mathbf{X}}\mathbf{F}$** is a **fourth-order tensor** satisfying:

$$d\mathbf{F} = \mathcal{D}_{\mathbf{X}}\mathbf{F} : d\mathbf{X}. \tag{24.85}$$

Regarding the Differentiation of the Matrix Inverse, for $\mathbf{F}(\mathbf{X}) = \mathbf{X}^{-1}$ we use the identity:

$$d(\mathbf{X}\mathbf{X}^{-1}) = 0 \Rightarrow d\mathbf{X}\mathbf{X}^{-1} + \mathbf{X}d\mathbf{X}^{-1} = 0. \tag{24.86}$$

Solving for $d\mathbf{X}^{-1}$:

$$d\mathbf{X}^{-1} = -\mathbf{X}^{-1}(d\mathbf{X})\mathbf{X}^{-1}. \tag{24.87}$$

Thus, the derivative is the **negative bilinear operator**:

$$\mathcal{D}_{\mathbf{X}}(\mathbf{X}^{-1}) = -(\mathbf{X}^{-1} \otimes \mathbf{X}^{-1}). \tag{24.88}$$

where $\otimes$ denotes the Kronecker product. For Differentiation of Tensor-Valued Functions. We need to have a differentiable tensor function $\mathcal{F} : \mathbb{R}^{m\times n\times p} \to \mathbb{R}^{a\times b\times c}$, the **Fréchet derivative** shall be a **higher-order tensor** $\mathcal{D}_{\mathcal{X}}\mathcal{F}$ satisfying:

$$d\mathcal{F} = \mathcal{D}_{\mathcal{X}}\mathcal{F} : d\mathcal{X}. \tag{24.89}$$

Let's do a Differentiation of Tensor Contraction. If $f(\mathcal{X}) = \mathcal{X} : \mathcal{A}$, where $\mathcal{X}, \mathcal{A}$ are second-order tensors, then:

$$\frac{\partial}{\partial \mathcal{X}}(\mathcal{X} : \mathcal{A}) = \mathcal{A}. \tag{24.90}$$

For a fourth-order tensor $\mathcal{C}$, if $f(\mathcal{X}) = \mathcal{C} : \mathcal{X}$, then:

$$\frac{\partial}{\partial \mathcal{X}}(\mathcal{C} : \mathcal{X}) = \mathcal{C}. \tag{24.91}$$

Differentiation can be also done in Non-Euclidean Spaces. For a manifold $\mathcal{M}$, differentiation is defined via **tangent spaces** $T_{\mathbf{X}}\mathcal{M}$, with the **covariant derivative** $\nabla_{\mathbf{X}}$ satisfying the **Levi-Civita connection**:

$$\nabla_{\mathbf{X}}\mathbf{Y} = \lim_{\epsilon\to 0} \frac{\mathrm{Proj}_{T_{\mathbf{X}+\epsilon\mathbf{H}}\mathcal{M}}(\mathbf{Y}(\mathbf{X}+\epsilon\mathbf{H})) - \mathbf{Y}(\mathbf{X})}{\epsilon}. \tag{24.92}$$

We can do differentiation using Variational Principles also. If $f(\mathbf{X})$ is an energy functional, the differentiation that follows from **Gateaux derivatives** is:

$$\delta f = \lim_{\epsilon\to 0} \frac{f(\mathbf{X}+\epsilon\mathbf{H}) - f(\mathbf{X})}{\epsilon}. \tag{24.93}$$

For **functionals**, differentiation uses **Euler-Lagrange equations**:

$$\frac{d}{dt}\int_{\Omega} L(\mathbf{X}, \nabla\mathbf{X})\, dV = 0. \tag{24.94}$$

24.5 Information Theory

Information Theory is a fundamental mathematical discipline concerned with the quantification, transmission, storage, and processing of information. It was first rigorously formulated by Claude Shannon in 1948 in his seminal paper *A Mathematical Theory of Communication*. The core idea is to measure the **amount of information** contained in a random process and determine how efficiently it can be encoded and transmitted through a noisy channel.

Formally, Information Theory is deeply intertwined with **probability theory, measure theory, functional analysis, and ergodic theory**, and it finds applications in diverse fields such as statistical mechanics, coding theory, artificial intelligence, and even quantum information.

24.5.1 Entropy: The Fundamental Measure of Uncertainty

Definition of Shannon Entropy: Let X be a discrete random variable taking values in a finite alphabet $\mathcal{X}$, with probability mass function (PMF) $p : \mathcal{X} \to [0, 1]$, satisfying

$$\sum_{x\in\mathcal{X}} p(x) = 1. \tag{24.95}$$

The Shannon **entropy** $H(X)$ is defined rigorously as the expected value of the logarithm of the inverse probability:

$$H(X) = \mathbb{E}[-\log p(X)] = -\sum_{x\in\mathcal{X}} p(x)\log p(x). \tag{24.96}$$

where the logarithm is taken in base 2 (bits) or natural base e (nats). Shannon's entropy satisfies the following fundamental properties, which are **derived axiomatically** using Khinchin's postulates:

1. **Continuity**: $H(X)$ is a continuous function of $p(x)$.
2. **Maximality**: The uniform distribution $p(x) = \frac{1}{n}$ for all $x \in \mathcal{X}$ maximizes entropy:
$$H(X) \leq \log n. \tag{24.97}$$
3. **Additivity**: For two independent random variables X and Y, entropy satisfies:
$$H(X,Y) = H(X) + H(Y). \tag{24.98}$$
4. **Monotonicity**: Conditioning reduces entropy:
$$H(X|Y) \leq H(X). \tag{24.99}$$

The Fundamental Theorem of Information Measures: Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, the Shannon entropy satisfies the variational principle:

$$H(X) = \inf_{Q} D_{\mathrm{KL}}(P\|Q), \tag{24.100}$$

where the infimum is taken over all probability measures Q on $\mathcal{X}$ and $D_{\mathrm{KL}}(P\|Q)$ is the **Kullback-Leibler divergence**:

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x} p(x)\log\frac{p(x)}{q(x)}. \tag{24.101}$$

Thus, entropy can be interpreted as the **minimum information divergence from uniformity**. Let $(\Omega, \mathcal{F}, P)$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is the σ-algebra of events, P is the probability measure. A discrete random variable X is a measurable function $X : \Omega \to \mathcal{X}$, where $\mathcal{X}$ is a countable set. The probability mass function (PMF) of X is given by:

$$p_X(x) = P(X = x) \tag{24.102}$$

The Shannon entropy of a discrete random variable X is defined as:

$$H(X) = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \tag{24.103}$$

where $0\log 0 \equiv 0$ by convention, and the logarithm is typically base 2 (bits) or base e (nats). For two random variables X and Y the joint entropy is:

$$H(X,Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{X,Y}(x,y) \tag{24.104}$$

The conditional entropy of Y given X is:

$$H(Y|X) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{Y|X}(y|x) \tag{24.105}$$

The mutual information between X and Y is:

$$I(X;Y) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)} \tag{24.106}$$

Regarding the **Non-Negativity of Entropy**, $H(X) \geq 0$, with equality if and only if X is deterministic. To prove this note that $p_X(x) \in [0,1]$, we have $-\log p_X(x) \geq 0$ for all $x \in \mathcal{X}$. Thus:

$$H(X) = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \geq 0 \tag{24.107}$$

Equality holds if and only if $p_X(x) = 1$ for some x and $p_X(x') = 0$ for all $x' \neq x$, meaning X is deterministic. To get an upper bound on Entropy, for a discrete random variable X with $|\mathcal{X}|$ possible outcomes:

$$H(X) \leq \log|\mathcal{X}| \tag{24.108}$$

with equality if and only if X is uniformly distributed. To prove this, using Gibbs' inequality, for any probability distributions $p_X(x)$ and $q_X(x)$:

$$-\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \leq -\sum_{x\in\mathcal{X}} p_X(x)\log q_X(x) \tag{24.109}$$

Let $q_X(x) = \frac{1}{|\mathcal{X}|}$ (uniform distribution). Then:

$$H(X) \leq -\sum_{x\in\mathcal{X}} p_X(x)\log\frac{1}{|\mathcal{X}|} = \log|\mathcal{X}| \tag{24.110}$$

Equality holds if and only if $p_X(x) = q_X(x) = \frac{1}{|\mathcal{X}|}$ for all x, meaning X is uniformly distributed. By chain Rule for Joint Entropy, for two random variables X and Y, the joint entropy satisfies:

$$H(X,Y) = H(X) + H(Y|X). \tag{24.111}$$

By definition:

$$H(X,Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{X,Y}(x,y). \tag{24.112}$$

Using the chain rule of probability, $p_{X,Y}(x,y) = p_X(x)p_{Y|X}(y|x)$, we rewrite:

$$H(X,Y) = -\sum_{x,y} p_{X,Y}(x,y)\log[p_X(x)p_{Y|X}(y|x)] \tag{24.113}$$

Splitting the logarithm:

$$H(X,Y) = -\sum_{x,y} p_{X,Y}(x,y)\log p_X(x) - \sum_{x,y} p_{X,Y}(x,y)\log p_{Y|X}(y|x). \tag{24.114}$$

The first term simplifies to $H(X)$, and the second term simplifies to $H(Y|X)$, giving:

$$H(X,Y) = H(X) + H(Y|X). \tag{24.115}$$

The mutual information $I(X;Y)$ satisfies:

$$I(X;Y) = H(X) + H(Y) - H(X,Y). \tag{24.116}$$

By definition:

$$I(X;Y) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}. \tag{24.117}$$

Using the definitions of entropy and joint entropy:

$$I(X;Y) = H(X) + H(Y) - H(X,Y). \tag{24.118}$$

We now discuss Mutual Information and Fundamental Theorems of Dependence. The **mutual information** between two random variables X and Y quantifies the reduction in uncertainty of X given knowledge of Y:

$$I(X;Y) = H(X) - H(X|Y). \tag{24.119}$$

Equivalently, it is given by the **relative entropy between the joint distribution** $p(x,y)$ **and the product of the marginals**:

$$I(X;Y) = D_{\mathrm{KL}}(p(x,y)\|p(x)p(y)). \tag{24.120}$$

For any Markov chain $X \to Y \to Z$, mutual information satisfies:

$$I(X;Z) \leq I(X;Y). \tag{24.121}$$

This follows directly from Jensen's inequality and the convexity of relative entropy.

24.5.2 Source Coding Theorem: Fundamental Limits of Compression

Given a source emitting i.i.d. symbols $X_1, X_2, \cdots \sim P_X$, the **Shannon Source Coding Theorem** states that for any uniquely decodable code, the expected length per symbol satisfies:

$$L \geq H(X). \tag{24.122}$$

Moreover, the **Asymptotic Equipartition Property (AEP)** states that for large n, the probability of a sequence $X_1, X_2, ..., X_n$ satisfies:

$$P(X_1, \ldots, X_n) \approx 2^{-nH(X)}. \tag{24.123}$$

The **Shannon Source Coding Theorem** states that:

1. **Achievability**: Given a discrete memoryless source (DMS) X with entropy $H(X)$, for any $\epsilon > 0$, there exists a source code that compresses sequences of length n to approximately $n(H(X) + \epsilon)$ bits per symbol and allows for decoding with vanishing error probability as $n \to \infty$.

2. **Converse**: No source code can achieve an average code length per symbol smaller than $H(X)$ without increasing the error probability to 1.

To prove the **Shannon Source Coding Theorem**, we assume that X is a discrete random variable with probability mass function (PMF) $P_X(x)$. The entropy of X is defined as:

$$H(X) = -\sum_{x\in\mathcal{X}} P_X(x) \log P_X(x). \tag{24.124}$$

For a sequence $X_1, X_2, \ldots, X_n$ drawn i.i.d. from P_X, the joint entropy satisfies:

$$H(X^n) = nH(X). \tag{24.125}$$

We will use the **Asymptotic Equipartition Property (AEP)**, which states that for large n, the sequence X^n belongs to a **typical set** $\mathcal{T}_\epsilon^{(n)}$ with high probability. The first step is to AEP and the Size of the Typical Set. The **strong law of large numbers** implies that for any $\epsilon > 0$, the set:

$$\mathcal{T}_\epsilon^{(n)} = \left\{ x^n \in \mathcal{X}^n : \left| -\frac{1}{n} \log P_X(x^n) - H(X) \right| < \epsilon \right\} \tag{24.126}$$

has probability approaching 1 as $n \to \infty$. Furthermore, the number of typical sequences satisfies:

$$\left|\mathcal{T}_\epsilon^{(n)}\right| \approx 2^{n(H(X)+\epsilon)}. \tag{24.127}$$

Since these sequences occur with high probability, we can restrict our coding efforts to them. The third step is to encode the Typical Sequences. If we assign a unique binary code to each sequence in $\mathcal{T}_\epsilon^{(n)}$, we need at most $\log |\mathcal{T}_\epsilon^{(n)}|$ bits per sequence, which gives an encoding length:

$$L \approx \log 2^{n(H(X)+\epsilon)} = n(H(X) + \epsilon). \tag{24.128}$$

Thus, the **expected code length per symbol is at most** $H(X)+\epsilon$. The third step is to analyze the Converse (Optimality of Entropy Rate). Consider any uniquely decodable prefix-free code with average code length L. By **Kraft's inequality**:

$$\sum_{x^n} 2^{-L(x^n)} \leq 1. \tag{24.129}$$

Taking logarithms and using Jensen's inequality, we obtain:

$$\mathbb{E}[L(X^n)] \geq H(X^n) = nH(X). \tag{24.130}$$

Thus, **no lossless source code can achieve a rate below** $H(X)$ **bits per symbol**. We have rigorously proven both the **achievability** and the **converse** of the Shannon Source Coding Theorem, showing that the fundamental limit of lossless compression is given by the entropy of the source.

To prove the **Asymptotic Equipartition Property (AEP)**, we assume that $(\Omega, \mathcal{F}, P)$ is a probability space, and let $\{X_i\}_{i=1}^{\infty}$ be a sequence of independent and identically distributed (i.i.d.) random variables defined on this space, taking values in a finite alphabet X. The joint distribution of the sequence $\mathbf{X}_n = (X_1, X_2, \ldots, X_n)$ is given by:

$$P_{\mathbf{X}_n}(x_n) = \prod_{i=1}^{n} P_X(x_i) \tag{24.131}$$

where P_X is the probability mass function (PMF) of X_i. The entropy of X, denoted $H(X)$, is defined as:

$$H(X) = -\sum_{x \in X} P_X(x) \log P_X(x) \tag{24.132}$$

where the logarithm is taken base 2, and $H(X)$ quantifies the expected information content of X. For a given $\epsilon > 0$ and sequence length n, the typical set $A_\epsilon(n)$ is defined as:

$$A_\epsilon(n) = \left\{ x_n \in X^n : \left| -\frac{1}{n} \log P_{\mathbf{X}_n}(x_n) - H(X) \right| < \epsilon \right\}. \tag{24.133}$$

This set consists of sequences x_n whose empirical entropy $-\frac{1}{n} \log P_{\mathbf{X}_n}(x_n)$ is close to the true entropy $H(X)$. The AEP states that, as $n \to \infty$, the probability of the typical set approaches 1:

$$\lim_{n\to\infty} P_{\mathbf{X}_n}(A_\epsilon(n)) = 1. \tag{24.134}$$

This is a direct consequence of the weak law of large numbers (WLLN) applied to the random variable $-\log P_X(X_i)$, which has finite mean $H(X)$ and finite variance (by the finiteness of X). The cardinality of the typical set satisfies:

$$(1-\epsilon)2^{n(H(X)-\epsilon)} \leq |A_\epsilon(n)| \leq 2^{n(H(X)+\epsilon)} \tag{24.135}$$

This follows from the definition of the typical set and the fact that $P_{\mathbf{X}_n}(x_n) \approx 2^{-nH(X)}$ for $x_n \in A_\epsilon(n)$. By Equipartition Property, we can say that for all $x_n \in A_\epsilon(n)$, the probability of x_n satisfies:

$$2^{-n(H(X)+\epsilon)} \leq P_{\mathbf{X}_n}(x_n) \leq 2^{-n(H(X)-\epsilon)}. \tag{24.136}$$

This implies that all sequences in the typical set are approximately equiprobable. The AEP is a consequence of the weak law of large numbers (WLLN) and the Chernoff bound. Here, we provide a rigorous proof. Define the random variable:

$$Y_i = -\log P_X(X_i). \tag{24.137}$$

Since $\{X_i\}$ are i.i.d., $\{Y_i\}$ are also i.i.d., with mean $E[Y_i] = H(X)$ and variance $\sigma^2 = \text{Var}(Y_i)$. By the Weak Law of Large Numbers, we can write:

$$\frac{1}{n}\sum_{i=1}^{n} Y_i \to_p H(X) \text{ as } n \to \infty. \tag{24.138}$$

This convergence in probability implies:

$$\lim_{n\to\infty} P\left(\left|-\frac{1}{n}\log P_{\mathbf{X}_n}(X_n) - H(X)\right| < \epsilon\right) = 1. \tag{24.139}$$

To quantify the rate of convergence, we use the Chernoff bound. For any $\epsilon > 0$, there exists $\delta > 0$ such that:

$$P\left(\left|-\frac{1}{n}\log P_{\mathbf{X}_n}(X_n) - H(X)\right| \geq \epsilon\right) \leq 2e^{-n\delta}. \tag{24.140}$$

This exponential decay ensures that the probability of non-typical sequences vanishes rapidly as $n \to \infty$. The AEP can be interpreted in the language of measure theory. The typical set $A_\epsilon(n)$ is a high-probability subset of X^n with respect to the product measure $P_{\mathbf{X}_n}$. The AEP asserts that, for large n, the measure $P_{\mathbf{X}_n}$ is concentrated on $A_\epsilon(n)$, and the uniform distribution on $A_\epsilon(n)$ approximates $P_{\mathbf{X}_n}$ in the sense of entropy. For a stationary and ergodic process $\{X_i\}$, the AEP holds with the entropy rate H replacing $H(X)$:

$$H = \lim_{n\to\infty} \frac{1}{n} H(X_n). \tag{24.141}$$

The typical set is defined analogously, and the probability concentration result holds under the ergodic theorem. For continuous random variables, the differential entropy $h(X)$ replaces $H(X)$, and the typical set is defined in terms of probability density functions:

$$A_\epsilon(n) = \left\{x_n \in \mathbb{R}^n : \left|-\frac{1}{n}\log f_{\mathbf{X}_n}(x_n) - h(X)\right| < \epsilon\right\}, \tag{24.142}$$

where $f_{\mathbf{X}_n}$ is the joint probability density function. For Markov chains and other non-i.i.d. processes, the AEP holds under appropriate mixing conditions, with the entropy rate adjusted to account for dependencies. The AEP underpins Shannon's source coding theorem, which states that the optimal compression rate for a source is given by its entropy rate.

24.5.3 Noisy Channel Coding Theorem: Fundamental Limits of Communication

Let X be the input and Y the output of a **discrete memoryless channel (DMC)** with transition probability $p(y|x)$. The **capacity** of the channel is given by:

$$C = \max_{p(x)} I(X;Y). \tag{24.143}$$

Shannon's **Noisy Channel Coding Theorem** asserts that for any transmission rate R:

- If $R \leq C$, there exists a code that allows error-free transmission.
- If $R > C$, error probability approaches 1.

For a discrete memoryless channel (DMC) with channel capacity C, there exists a coding scheme such that for any rate $R < C$ and any $\epsilon > 0$, there is a block code of length n and rate R with a decoding error probability $P_e \leq \epsilon$. Conversely, for any rate $R > C$, reliable communication is impossible. To prove this, we define $(X, Y, P_{Y|X})$ as the DMC, where X is the input alphabet, Y is the output alphabet, $P_{Y|X}(y|x)$ is the conditional probability distribution characterizing the channel. The channel is memoryless, meaning:

$$P_{Y^n|X^n}(y|x) = \prod_{i=1}^{n} P_{Y|X}(y_i|x_i) \tag{24.144}$$

The channel capacity C is defined as:

$$C = \max_{P_X} I(X;Y), \tag{24.145}$$

where $I(X;Y)$ is the mutual information between X and Y, and the maximization is over all input distributions P_X.

Fix a rate $R < C$ and a small $\epsilon > 0$. By Random Coding Argument, Consider a block code of length n with $M = 2^{nR}$ codewords. Each codeword $x_m = (x_{m1}, x_{m2}, \ldots, x_{mn})$ is generated independently and identically according to the input distribution P_X that achieves capacity. **Encoding** means to transmit message m, send codeword x_m and **Decoding** means that upon receiving y, the decoder uses joint typicality decoding. Decode y to $\hat{m}$ if $(x_{\hat{m}}, y)$ are jointly typical and no other codeword is jointly typical with y. If no such $\hat{m}$ exists or multiple exist, declare an error. Regarding the Joint Typicality, the set of jointly typical sequences $A_\epsilon^{(n)}$ is defined as:

$$A_\epsilon^{(n)} = \left\{ (x, y) \in X^n \times Y^n : \left| -\frac{1}{n} \log P_{X^n,Y^n}(x, y) - H(X, Y) \right| < \epsilon \right\} \tag{24.146}$$

where $H(X, Y)$ is the joint entropy of X and Y. By the Joint Asymptotic Equipartition Property (AEP), for sufficiently large n:

$$P_{X^n,Y^n}(A_\epsilon^{(n)}) \geq 1 - \epsilon. \tag{24.147}$$

Doing the Error Probability Analysis, the error probability P_e is decomposed into two events:

- E_1: $(x_m, y) \notin A_\epsilon^{(n)}$.
- E_2: Some other codeword $x_{m'}$ (with $m' \neq m$) satisfies $(x_{m'}, y) \in A_\epsilon^{(n)}$.

Bounding $P(E_1)$, By the Joint AEP:

$$P(E_1) = P\left((x_m, y) \notin A_\epsilon^{(n)}\right) \leq \epsilon. \tag{24.148}$$

Bounding $P(E_2)$, for a fixed incorrect codeword $x_{m'}$, the probability that $(x_{m'}, y) \in A_\epsilon^{(n)}$ is approximately $2^{-nI(X;Y)}$. Since there are $M - 1 \approx 2^{nR}$ incorrect codewords, the union bound gives:

$$P(E_2) \leq (M - 1) \cdot 2^{-nI(X;Y)} \leq 2^{nR} \cdot 2^{-nI(X;Y)} = 2^{-n(I(X;Y)-R)}. \tag{24.149}$$

Since $R < C = I(X;Y)$, $P(E_2) \to 0$ exponentially as $n \to \infty$. Combining the bounds to get the total Error Probability:

$$P_e \leq P(E_1) + P(E_2) \leq \epsilon + 2^{-n(I(X;Y)-R)}. \tag{24.150}$$

For sufficiently large n, $P_e \leq 2\epsilon$. The converse part shows that reliable communication is impossible for $R > C$. The key steps are:

- Use Fano's inequality to relate the error probability P_e to the conditional entropy $H(M|\hat{M})$.
- Apply the data processing inequality to bound the mutual information $I(M;\hat{M})$.
- Show that if $R > C$, the error probability P_e cannot vanish.

Taking Measure-Theoretic Considerations, the proof assumes finite alphabets X and Y. For continuous alphabets, the same ideas apply, but integrals replace sums, and differential entropy replaces discrete entropy. The existence of the capacity-achieving input distribution P_X is guaranteed by the continuity and compactness of the mutual information functional. Regarding Asymptotic Analysis, The error probability P_e decays exponentially with n for $R < C$, as shown by the term $2^{-n(I(X;Y)-R)}$. This exponential decay is a consequence of the law of large numbers and the Chernoff bound. For any $R < C$ and $\epsilon > 0$, there exists a code of rate R with error probability $P_e \leq \epsilon$. Conversely, for $R > C$, reliable communication is impossible. This completes the rigorous proof of the Noisy Channel Coding Theorem.

24.5.4 Rate-Distortion Theory: Lossy Data Compression

For a source X reconstructed as $\hat{X}$, the **rate-distortion function** determines the minimum achievable rate $R(D)$ for a given distortion D:

$$R(D) = \min_{p(\hat{x}|x):\mathbb{E}[d(X,\hat{X})]\leq D} I(X;\hat{X}). \tag{24.151}$$

Let X be a random variable representing the source data, with probability distribution $p_X(x)$ defined over a finite alphabet $\mathcal{X}$. The compressed representation of X is denoted by $\hat{X}$, which takes values in a finite alphabet $\hat{\mathcal{X}}$. The distortion between X and $\hat{X}$ is quantified by a distortion measure $d : \mathcal{X} \times \hat{\mathcal{X}} \to \mathbb{R}_{\geq 0}$, which is assumed to be non-negative and bounded. The Rate-Distortion Function $R(D)$ is defined as:

$$R(D) = \inf_{p_{\hat{X}|X}} \left\{ I(X;\hat{X}) : E[d(X,\hat{X})] \leq D \right\} \tag{24.152}$$

where $p_{\hat{X}|X}$ is the conditional distribution of $\hat{X}$ given X, $I(X;\hat{X})$ is the mutual information between X and $\hat{X}$, $E[d(X,\hat{X})]$ is the expected distortion. The infimum is taken over all conditional distributions $p_{\hat{X}|X}$ that satisfy the distortion constraint $E[d(X,\hat{X})] \leq D$. The mutual information $I(X;\hat{X})$ is defined as:

$$I(X;\hat{X}) = \sum_{x\in\mathcal{X}} \sum_{\hat{x}\in\hat{\mathcal{X}}} p_X(x) p_{\hat{X}|X}(\hat{x}|x) \log \frac{p_{\hat{X}|X}(\hat{x}|x)}{p_{\hat{X}}(\hat{x})} \tag{24.153}$$

where $p_{\hat{X}}(\hat{x}) = \sum_{x\in\mathcal{X}} p_X(x) p_{\hat{X}|X}(\hat{x}|x)$ is the marginal distribution of $\hat{X}$. The expected distortion is given by:

$$E[d(X,\hat{X})] = \sum_{x\in\mathcal{X}} \sum_{\hat{x}\in\hat{\mathcal{X}}} p_X(x) p_{\hat{X}|X}(\hat{x}|x) d(x,\hat{x}) \tag{24.154}$$

The problem of finding $R(D)$ is a constrained optimization problem:

$$\text{Minimize } I(X;\hat{X}) \text{ subject to } E[d(X,\hat{X})] \leq D \tag{24.155}$$

This is a convex optimization problem because:

- The mutual information $I(X;\hat{X})$ is a convex function of $p_{\hat{X}|X}$,
- The distortion constraint $E[d(X,\hat{X})] \leq D$ is a linear (and thus convex) constraint.

We now give the Proof of the Rate-Distortion Function. To prove the convexity of $R(D)$, consider two distortion levels D_1 and D_2, and let p_1 and p_2 be the corresponding optimal conditional distributions achieving $R(D_1)$ and $R(D_2)$, respectively. For any $\lambda \in [0,1]$, define:

$$D_\lambda = \lambda D_1 + (1-\lambda) D_2 \tag{24.156}$$

The conditional distribution $p_\lambda = \lambda p_1 + (1-\lambda) p_2$ achieves an expected distortion of D_λ. By the convexity of mutual information:

$$I(X;\hat{X}) \leq \lambda I(X;\hat{X}_1) + (1-\lambda) I(X;\hat{X}_2) \tag{24.157}$$

Thus:

$$R(D_\lambda) \leq \lambda R(D_1) + (1-\lambda) R(D_2) \tag{24.158}$$

proving the convexity of $R(D)$. Regarding the Monotonicity of $R(D)$, The Rate-Distortion Function $R(D)$ is non-increasing in D. Formally, if $D_1 \leq D_2$, then:

$$R(D_1) \geq R(D_2) \tag{24.159}$$

This follows because the set of conditional distributions $p_{\hat{X}|X}$ satisfying $E[d(X,\hat{X})] \leq D_2$ includes all distributions satisfying $E[d(X,\hat{X})] \leq D_1$.

The achievability of $R(D)$ is proven using the random coding argument. For a given D, generate a codebook of 2^{nR} codewords, each drawn independently according to the marginal distribution $p_{\hat{X}}(\hat{x})$. For each source sequence x^n, find the codeword $\hat{x}^n$ that minimizes the distortion $d(x^n, \hat{x}^n)$. Using the law of large numbers and the typicality of sequences, it can be shown that the expected distortion approaches D as the block length $n \to \infty$, provided $R \geq R(D)$. The converse is proven using the data processing inequality and the properties of mutual information. Suppose there exists a code with rate $R < R(D)$ and distortion $E[d(X,\hat{X})] \leq D$. Then:

$$R \geq I(X;\hat{X}) \geq R(D) \tag{24.160}$$

which is a contradiction. Thus, $R(D)$ is the fundamental limit. The optimization problem can be reformulated using the Lagrangian:

$$L(p_{\hat{X}|X}, \lambda) = I(X;\hat{X}) + \lambda \Big(E[d(X,\hat{X})] - D \Big) \tag{24.161}$$

where $\lambda \geq 0$ is the Lagrange multiplier. The optimal solution satisfies the Karush-Kuhn-Tucker (KKT) conditions:

1. Stationarity:
$$\nabla_{p_{\hat{X}|X}} L = 0. \tag{24.162}$$

2. Primal Feasibility:
$$E[d(X,\hat{X})] \leq D. \tag{24.163}$$

3. Dual Feasibility:
$$\lambda \geq 0. \tag{24.164}$$

4. Complementary Slackness:
$$\lambda \Big(E[d(X,\hat{X})] - D \Big) = 0. \tag{24.165}$$

The Blahut-Arimoto algorithm is an iterative method for numerically computing $R(D)$. It alternates between updating the conditional distribution $p_{\hat{X}|X}$ and the Lagrange multiplier λ to converge to the optimal solution. For a Gaussian source $X \sim N(0, \sigma^2)$ and squared-error distortion $d(x, \hat{x}) = (x - \hat{x})^2$, the Rate-Distortion Function is:

$$R(D) = \begin{cases} \frac{1}{2} \log_2 \left(\frac{\sigma^2}{D} \right), & 0 \leq D \leq \sigma^2, \\ 0, & D > \sigma^2. \end{cases} \tag{24.166}$$

This result is derived using the properties of Gaussian distributions and mutual information, and it illustrates the trade-off between rate and distortion. The Rate-Distortion Function $R(D)$ is a cornerstone of information theory, rigorously characterizing the fundamental limits of lossy data compression. This deep theoretical framework underpins modern data compression techniques and has broad applications in communication, signal processing, and machine learning.

24.5.5 Applications of Information Theory

There are several applications of Information Theory:

Error-Correcting Codes: Reed-Solomon, Turbo, and LDPC codes achieve rates near capacity. The channel capacity C is the supremum of all achievable rates R for which there exists a coding scheme with a vanishing probability of error $P_e \to 0$ as the block length $n \to \infty$. For a discrete memoryless channel (DMC) with transition probabilities $P(y|x)$, the capacity is given by:

$$C = \sup_{P_X} I(X; Y) \tag{24.167}$$

where $I(X; Y)$ is the mutual information between the input X and output Y, and the supremum is taken over all input distributions P_X. For the additive white Gaussian noise (AWGN) channel with power constraint P and noise variance σ^2, the capacity is:

$$C = \frac{1}{2} \log_2 \left(1 + \frac{P}{\sigma^2} \right) \quad \text{[bits per channel use]}. \tag{24.168}$$

The converse of Shannon's theorem establishes that no coding scheme can achieve $R > C$ with $P_e \to 0$. Let's now discuss the Fundamental Limits and Large Deviation Theory of Error-Correcting Codes. An error-correcting code C of block length n and rate $R = k/n$ maps k information bits to n coded bits. The error exponent $E(R)$ characterizes the exponential decay of P_e with n for rates $R < C$:

$$P_e \sim e^{-nE(R)}. \tag{24.169}$$

The Gallager exponent provides a lower bound on $E(R)$:

$$E(R) = \max_{0 \leq \rho \leq 1} \left[E_0(\rho) - \rho R \right], \tag{24.170}$$

where $E_0(\rho)$ is the Gallager function:

$$E_0(\rho) = -\log_2 \left(\sum_y \left(\sum_x P_X(x) P(y|x)^{\frac{1}{1+\rho}} \right)^{1+\rho} \right). \tag{24.171}$$

For the AWGN channel, $E_0(\rho)$ can be expressed in terms of the signal-to-noise ratio (SNR). Let's discuss the Algebraic Geometry and Finite Fields of Reed-Solomon Codes. Reed-Solomon codes are evaluation codes defined over finite fields $\mathbb{F}_q$, where $q = 2^m$. They are constructed by evaluating polynomials of degree $k-1$ at distinct points $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$. For encoding, The message polynomial $m(x) \in \mathbb{F}_q[x]$ of degree $k-1$ is encoded into a codeword:

$$c = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n)). \tag{24.172}$$

For Decoding, The Berlekamp-Welch algorithm or Guruswami-Sudan algorithm is used to correct up to $t = \lfloor (n-k)/2 \rfloor$ errors. The latter achieves list decoding, allowing correction of up to $n - \frac{n}{k}$ errors. The Weil conjectures and Riemann-Roch theorem provide deep insights into the algebraic structure of Reed-Solomon codes and their generalizations, such as algebraic geometry codes.

Regarding Turbo Codes: Iterative Decoding and Statistical Mechanics. Turbo codes are constructed using two recursive systematic convolutional (RSC) encoders separated by an interleaver. The iterative decoding process can be analyzed using tools from statistical mechanics.

1. **Factor Graph Representation:** The decoding process is represented as message passing on a factor graph, where the nodes correspond to variables and constraints. The Bethe free energy provides a variational characterization of the decoding problem.

2. **EXIT Charts:** The extrinsic information transfer (EXIT) chart is a tool to analyze the convergence of iterative decoding. The area theorem relates the area under the EXIT curve to the gap to capacity.

The performance of Turbo codes is characterized by the waterfall region and the error floor, which can be analyzed using large deviation theory and random matrix theory. LDPC codes are defined by a sparse parity-check matrix $H \in \mathbb{F}_2^{m \times n}$, where each row represents a parity-check constraint. The Tanner graph of the code is a bipartite graph with variable nodes (corresponding to codeword bits) and check nodes (corresponding to parity constraints). Regarding the Message-Passing Decoding, The sum-product algorithm (SPA) or min-sum algorithm (MSA) is used for iterative decoding. The messages passed between nodes are log-likelihood ratios (LLRs). Regarding the Density Evolution, This is a theoretical tool to analyze the asymptotic performance of LDPC codes. It tracks the probability density function (PDF) of the LLRs as a function of the iteration number. The threshold of the code is the maximum noise level for which $P_e \to 0$ as $n \to \infty$. The degree distributions of the variable and check nodes, denoted by $\lambda(x)$ and $\rho(x)$, respectively, are optimized to maximize the threshold. The optimization problem can be formulated as:

$$\max_{\lambda, \rho} \text{Threshold}(\lambda, \rho) \quad \text{subject to} \quad \int_0^1 \lambda(x) dx = \int_0^1 \rho(x) dx = 1. \tag{24.173}$$

The near-capacity performance of Turbo and LDPC codes is a consequence of their ability to exploit the channel's soft information and their iterative decoding algorithms. The turbo principle states that the exchange of extrinsic information between decoders improves the reliability of the estimates.

Machine Learning: KL-divergence and mutual information are used in variational inference. We begin by placing the problem in a measure-theoretic framework. Let $(\Omega, \mathcal{F}, P)$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is a σ-algebra, and P is a probability measure. The observed variables x and latent variables z are random variables defined on this space, with

$$x : \Omega \to X, \quad z : \Omega \to Z, \tag{24.174}$$

where X and Z are measurable spaces. The joint distribution $p(x, z)$ is a probability measure on $X \times Z$, and the posterior $p(z \mid x)$ is a conditional probability measure. Variational inference seeks to approximate $p(z \mid x)$ using a variational measure $q(z; \phi)$, where ϕ parameterizes the variational family $\mathcal{Q}$. The Kullback-Leibler (KL) divergence between two probability measures Q and P on $(Z, \mathcal{G})$ is defined as:

$$D_{\mathrm{KL}}(Q \parallel P) = \int_Z \log\left(\frac{dQ}{dP}\right) dQ, \tag{24.175}$$

where $\frac{dQ}{dP}$ is the Radon-Nikodym derivative of Q with respect to P. The KL divergence is finite only if Q is absolutely continuous with respect to P (denoted $Q \ll P$), and it satisfies:

$$D_{\mathrm{KL}}(Q \parallel P) \geq 0, \tag{24.176}$$

$$D_{\mathrm{KL}}(Q \parallel P) = 0 \quad \text{if and only if} \quad Q = P \text{ almost everywhere}. \tag{24.177}$$

In variational inference (VI), $Q = q(z; \phi)$ and $P = p(z \mid x)$, and we minimize $D_{\mathrm{KL}}(q(z; \phi) \parallel p(z \mid x))$. Variational Inference can be viewed as an optimization problem in a function space. Let $\mathcal{Q}$ be a family of probability measures on Z, and define the functional:

$$F[q] = D_{\mathrm{KL}}(q(z; \phi) \parallel p(z \mid x)) \tag{24.178}$$

The goal is to find:

$$q^* = \arg\min_{q \in \mathcal{Q}} F[q] \tag{24.179}$$

This is a constrained optimization problem, where q must satisfy:

$$\int_Z q(z; \phi)\, dz = 1, \quad q(z; \phi) \geq 0. \tag{24.180}$$

The Evidence Lower Bound (ELBO) is derived using measure-theoretic expectations. Starting from the log-marginal likelihood:

$$\log p(x) = \log \int_Z p(x, z)\, dz \tag{24.181}$$

we introduce $q(z; \phi)$ and apply Jensen's inequality:

$$\log p(x) \geq \int_Z q(z; \phi) \log \frac{p(x, z)}{q(z; \phi)}\, dz \equiv \mathrm{ELBO}(\phi) \tag{24.182}$$

The ELBO can be expressed as:

$$\mathrm{ELBO}(\phi) = \mathbb{E}_{q(z;\phi)}[\log p(x, z)] + H[q(z; \phi)] \tag{24.183}$$

where

$$H[q(z; \phi)] = -\mathbb{E}_{q(z;\phi)}[\log q(z; \phi)] \tag{24.184}$$

is the entropy of $q(z; \phi)$. The mutual information between x and z is defined as:

$$I(x; z) = D_{\mathrm{KL}}(p(x, z) \parallel p(x) \otimes p(z)), \tag{24.185}$$

where $p(x) \otimes p(z)$ is the product measure of the marginals. In VI, the variational mutual information is:

$$I_q(x; z) = \mathbb{E}_{p(x)}\left[D_{\mathrm{KL}}(q(z \mid x) \parallel q(z))\right] \tag{24.186}$$

where

$$q(z) = \int_X q(z \mid x) p(x)\, dx \tag{24.187}$$

is the aggregated posterior. Using measure-theoretic expectations, the ELBO can be decomposed as:

$$\mathrm{ELBO}(\phi) = \mathbb{E}_{p(x)}\left[\mathbb{E}_{q(z|x)}[\log p(x \mid z)]\right] - I_q(x; z) - D_{\mathrm{KL}}(q(z) \parallel p(z)). \tag{24.188}$$

Quantum Information: von Neumann entropy generalizes Shannon entropy for quantum states. In quantum mechanics, the state of a quantum system is described by a density operator ρ, which is a positive semi-definite, Hermitian operator acting on a Hilbert space $\mathcal{H}$, with unit trace:

$$\rho \geq 0, \quad \rho = \rho^\dagger, \quad \mathrm{Tr}(\rho) = 1. \tag{24.189}$$

For a pure state $|\psi\rangle \in \mathcal{H}$, the density operator is given by:

$$\rho = |\psi\rangle \langle\psi| . \tag{24.190}$$

For a mixed state, which is a statistical ensemble of pure states $\{|\psi_i\rangle\}$ with probabilities $\{p_i\}$, the density operator is:

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| . \tag{24.191}$$

The spectral theorem guarantees that any density operator ρ can be diagonalized in terms of its eigenvalues $\{\lambda_i\}$ and eigenstates $\{|\phi_i\rangle\}$:

$$\rho = \sum_i \lambda_i |\phi_i\rangle \langle\phi_i| , \tag{24.192}$$

where $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, and $\{|\phi_i\rangle\}$ forms an orthonormal basis for $\mathcal{H}$. We first give the definition and functional calculus of Von Neumann Entropy. The von Neumann entropy $S(\rho)$ of a quantum state ρ is defined as:

$$S(\rho) = -\mathrm{Tr}(\rho \log \rho). \tag{24.193}$$

Since ρ is a positive semi-definite operator, the logarithm of ρ is defined via its spectral decomposition. If

$$\rho = \sum_i \lambda_i |\phi_i\rangle \langle\phi_i| , \tag{24.194}$$

then:

$$\log \rho = \sum_i \log \lambda_i |\phi_i\rangle \langle\phi_i| . \tag{24.195}$$

Here, $\log \lambda_i$ is well-defined for $\lambda_i > 0$. By convention,

$$0 \log 0 = 0, \tag{24.196}$$

which is consistent with the limit $\lim\limits_{x\to 0^+} x \log x = 0$. The trace operation is linear and invariant under cyclic permutations. Using the spectral decomposition of ρ, we have:

$$S(\rho) = -\mathrm{Tr}\left(\sum_i \lambda_i |\phi_i\rangle \langle\phi_i| \cdot \sum_j \log \lambda_j |\phi_j\rangle \langle\phi_j| \right) . \tag{24.197}$$

Simplifying this expression using the orthonormality of $\{|\phi_i\rangle\}$, we obtain:

$$S(\rho) = -\sum_i \lambda_i \log \lambda_i. \tag{24.198}$$

This is the quantum analog of the Shannon entropy, where the eigenvalues $\{\lambda_i\}$ of ρ play the role of classical probabilities. There are many Mathematical Properties of Von Neumann Entropy. The first of them is **Non-negativity**:

$$S(\rho) \geq 0, \tag{24.199}$$

with equality if and only if ρ is a pure state (i.e., $\rho = |\psi\rangle \langle\psi|$ for some $|\psi\rangle$). For a d-dimensional Hilbert space $\mathcal{H}$, the von Neumann entropy is maximized by the maximally mixed state $\rho = \frac{I}{d}$, where I is the identity operator on $\mathcal{H}$. The maximum entropy is:

$$S\left(\frac{I}{d}\right) = \log d. \tag{24.200}$$

The von Neumann entropy is concave in ρ. For any set of density operators $\{\rho_i\}$ and probabilities $\{p_i\}$, we have:

$$S\left(\sum_i p_i\rho_i\right) \geq \sum_i p_i S(\rho_i). \tag{24.201}$$

This reflects the fact that mixing quantum states increases uncertainty. For a composite system described by a product state $\rho_{AB} = \rho_A \otimes \rho_B$, the entropy is additive:

$$S(\rho_{AB}) = S(\rho_A) + S(\rho_B). \tag{24.202}$$

Physics: Maximum entropy methods are foundational in statistical mechanics. The maximum entropy principle is a variational principle that selects the probability distribution $\{p_i\}$ over microstates i of a system by maximizing the Shannon entropy functional $S[p]$, subject to a set of constraints that encode known macroscopic information about the system. Regarding the Shannon Entropy Functional, for a discrete probability distribution $\{p_i\}$, the Shannon entropy is defined as:

$$S[p] = -k_B \sum_{i\in M} p_i \ln p_i \tag{24.203}$$

where M is the set of all microstates of the system, k_B is the Boltzmann constant, which ensures dimensional consistency with thermodynamic entropy, p_i is the probability of the system being in microstate i, satisfying $p_i \geq 0$ and $\sum_i p_i = 1$. For a continuous probability distribution $p(x)$ over a state space X, the entropy is defined as:

$$S[p] = -k_B \int_X p(x) \ln p(x)\, dx \tag{24.204}$$

where $p(x)$ is a probability density function (PDF) satisfying $p(x) \geq 0$ and $\int_X p(x)\, dx = 1$. In this problem, Constraints and Macroscopic Observables, The system is subject to a set of m macroscopic constraints, which are expressed as expectation values of observables $\{A_k\}_{k=1}^m$. These constraints take the form:

$$\langle A_k\rangle = \sum_{i\in M} p_i A_k(i) = a_k, \quad k = 1, 2, \ldots, m \tag{24.205}$$

where $A_k(i)$ is the value of the observable A_k in microstate i, and a_k is the measured or expected value of A_k. The normalization constraint $\sum_i p_i = 1$ is always included. We have to now setup the Variational Formulation and Lagrange Multipliers. The constrained optimization problem is formulated using the method of Lagrange multipliers. We define the Lagrangian functional:

$$L[p, \{\lambda_k\}] = S[p] - \lambda_0 \left(\sum_i p_i - 1\right) - \sum_{k=1}^m \lambda_k \left(\sum_i p_i A_k(i) - a_k\right) \tag{24.206}$$

where λ_0 is the Lagrange multiplier for the normalization constraint, λ_k are the Lagrange multipliers for the macroscopic constraints. Regarding the Functional Derivative and Stationarity Condition, To find the extremum of L, we take the functional derivative of L with respect to p_i and set it to zero:

$$\frac{\delta L}{\delta p_i} = -k_B(\ln p_i + 1) - \lambda_0 - \sum_{k=1}^m \lambda_k A_k(i) = 0 \tag{24.207}$$

Solving for p_i:

$$\ln p_i = -\frac{1+\lambda_0}{k_B} - \sum_{k=1}^m \frac{\lambda_k}{k_B} A_k(i) \tag{24.208}$$

Exponentiating both sides:

$$p_i = \exp\left(-\frac{1+\lambda_0}{k_B} - \sum_{k=1}^m \frac{\lambda_k}{k_B} A_k(i)\right) \tag{24.209}$$

Let $Z = \exp\left(\frac{1+\lambda_0}{k_B}\right)$, which acts as a normalization constant (partition function). Then:

$$p_i = \frac{1}{Z}\exp\left(-\sum_{k=1}^{m}\frac{\lambda_k}{k_B}A_k(i)\right) \tag{24.210}$$

Regarding the Identification of Lagrange Multipliers, The Lagrange multipliers $\{\lambda_k\}$ are determined by enforcing the constraints. For example: If $A_1(i) = E_i$ (energy of microstate i), then $\lambda_1 = \beta = \frac{1}{k_B T}$, where T is the temperature and If

$$A_2(i) = N_i \tag{24.211}$$

(particle number in microstate i), then

$$\lambda_2 = -\beta\mu, \tag{24.212}$$

where μ is the chemical potential. The resulting probability distribution is:

$$p_i = \frac{1}{Z}\exp(-\beta E_i + \beta\mu N_i), \tag{24.213}$$

which is the grand canonical distribution. The entropy functional $S[p]$ is strictly concave in p, and the constraints are linear in p. By the properties of convex optimization:

- The solution to the constrained optimization problem exists and is unique.
- The maximum entropy distribution is the unique global maximizer of $S[p]$ subject to the constraints.

The maximum entropy principle is deeply connected to thermodynamics through the following relationships. The partition function Z is given by:

$$Z = \sum_i \exp(-\beta E_i + \beta\mu N_i). \tag{24.214}$$

The free energy F is related to Z by:

$$F = -k_B T \ln Z. \tag{24.215}$$

The entropy S and expected energy $\langle E\rangle$ are:

$$S = k_B(\ln Z + \beta\langle E\rangle) \tag{24.216}$$

$$\langle E\rangle = -\frac{\partial \ln Z}{\partial \beta} \tag{24.217}$$

The maximum entropy principle naturally leads to the identification of thermodynamic potentials, such as the Helmholtz free energy F, Gibbs free energy G, and grand potential Φ. The maximum entropy distribution can be derived from large deviation theory, which describes the exponential decay of probabilities of rare events. The Boltzmann distribution emerges as the most probable macrostate in the thermodynamic limit. The space of probability distributions equipped with the Fisher information metric forms a Riemannian manifold. The maximum entropy principle corresponds to finding the distribution closest to the uniform distribution (maximum ignorance) in this geometric framework. For non-equilibrium systems, the maximum entropy principle can be extended using relative entropy (Kullback-Leibler divergence) or dynamical constraints, such as fixed currents or fluxes. The maximum entropy principle is rigorously justified by:

- **Sanov's Theorem:** A result in large deviation theory that characterizes the probability of observing an empirical distribution deviating from the true distribution.

- **Gibbs' Inequality:** The Shannon entropy is maximized by the uniform distribution when no constraints are imposed.

- **Convex Duality:** The Lagrange multipliers $\{\lambda_k\}$ are dual variables that encode the sensitivity of the entropy to changes in the constraints.

There are many applications of the maximum entropy principle in statistical mechanics. The maximum entropy principle is used to derive:

- The Boltzmann distribution for the canonical ensemble.

- The Fermi-Dirac and Bose-Einstein distributions for quantum systems.

- The Gibbs distribution for systems with multiple conserved quantities.

While the maximum entropy principle is powerful, it has limitations:

- It assumes knowledge of the correct constraints.

- It may not apply to systems with long-range correlations or non-Markovian dynamics.

- Extensions to non-equilibrium systems remain an active area of research.

In summary, the maximum entropy methods in statistical mechanics are a rigorous and foundational framework for inferring probability distributions based on limited information. They are deeply rooted in information theory, convex optimization, and statistical physics, and they provide a profound connection between microscopic dynamics and macroscopic thermodynamics.

24.5.6 Conclusion: Information Theory as a Universal Mathematical Principle

Information Theory provides a **rigorous mathematical framework** for encoding, transmission, and processing of information. Its deep connections to probability, optimization, and functional analysis make it central to digital communication, data science, and beyond.

25 Acknowledgments

The authors acknowledge the contributions of researchers whose foundational work has shaped our understanding of Deep Learning.

Bibliography

[1] Rao, N., Farid, M., and Raiz, M. (2024). Symmetric Properties of λ-Szász Operators Coupled with Generalized Beta Functions and Approximation Theory. Symmetry, 16(12), 1703.

[2] Mukhopadhyay, S.N., Ray, S. (2025). Function Spaces. In: Measure and Integration. University Texts in the Mathematical Sciences. Springer, Singapore.

[3] Szołdra, T. (2024). Ergodicity breaking in quantum systems: from exact time evolution to machine learning (Doctoral dissertation).

[4] SONG, W. X., CHEN, H., CUI, C., LIU, Y. F., TONG, D., GUO, F., ... and XIAO, C. W. (2025). Theoretical, methodological, and implementation considerations for establishing a sustainable urban renewal model. JOURNAL OF NATURAL RESOURCES, 40(1), 20-38.

[5] El Mennaoui, O., Kharou, Y., and Laasri, H. (2025). Evolution families in the framework of maximal regularity. Evolution Equations and Control Theory, 0-0.

[6] Pedroza, G. (2024). On the Conditions for Domain Stability for Machine Learning: a Mathematical Approach. arXiv preprint arXiv:2412.00464.

[7] Cerreia-Vioglio, S., and Ok, E. A. (2024). Abstract integration of set-valued functions. Journal of Mathematical Analysis and Applications, 129169.

[8] Averin, A. (2024). Formulation and Proof of the Gravitational Entropy Bound. arXiv preprint arXiv:2412.02470.

[9] Potter, T. (2025). Subspaces of $L^2(\mathbb{R}^n)$ Invariant Under Crystallographic Shifts. arXiv e-prints, arXiv-2501.

[10] Lee, M. (2025). Emergence of Self-Identity in Artificial Intelligence: A Mathematical Framework and Empirical Study with Generative Large Language Models. Axioms, 14(1), 44.

[11] Wang, R., Cai, L., Wu, Q., and Niyato, D. (2025). Service Function Chain Deployment with Intrinsic Dynamic Defense Capability. IEEE Transactions on Mobile Computing.

[12] Duim, J. L., and Mesquita, D. P. (2025). Artificial Intelligence Value Alignment via Inverse Reinforcement Learning. Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, 11(1), 1-2.

[13] Khayat, M., Barka, E., Serhani, M. A., Sallabi, F., Shuaib, K., and Khater, H. M. (2025). Empowering Security Operation Center with Artificial Intelligence and Machine Learning–A Systematic Literature Review. IEEE Access.

[14] Agrawal, R. (2025). 46 Detection of melanoma using DenseNet-based adaptive weighted loss function. Emerging Trends in Computer Science and Its Application, 283.

[15] Hailemichael, H., and Ayalew, B. Adaptive and Safe Fast Charging of Lithium-Ion Batteries Via Hybrid Model Learning and Control Barrier Functions. Available at SSRN 5110597.

[16] Nguyen, E., Xiao, J., Fan, Z., and Ruan, D. Contrast-free Full Intracranial Vessel Geometry Estimation from MRI with Metric Learning based Inference. In Medical Imaging with Deep Learning.

[17] Luo, Z., Bi, Y., Yang, X., Li, Y., Wang, S., and Ye, Q. A Novel Machine Vision-Based Collision Risk Warning Method for Unsignalized Intersections on Arterial Roads. Frontiers in Physics, 13, 1527956.

[18] Bousquet, N., Thomassé, S. (2015). VC-dimension and Erdős–Pósa property. Discrete Mathematics, 338(12), 2302-2317.

[19] Asian, O., Yildiz, O. T., Alpaydin, E. (2009, September). Calculating the VC-dimension of decision trees. In 2009 24th International Symposium on Computer and Information Sciences (pp. 193-198). IEEE.

[20] Zhang, C., Bian, W., Tao, D., Lin, W. (2012). Discretized-Vapnik-Chervonenkis dimension for analyzing complexity of real function classes. IEEE transactions on neural networks and learning systems, 23(9), 1461-1472.

[21] Riondato, M., Akdere, M., Çetintemel, U., Zdonik, S. B., Upfal, E. (2011). The VC-dimension of SQL queries and selectivity estimation through sampling. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II 22 (pp. 661-676). Springer Berlin Heidelberg.

[22] Bane, M., Riggle, J., Sonderegger, M. (2010). The VC dimension of constraint-based grammars. Lingua, 120(5), 1194-1208.

[23] Anderson, A. (2023). Fuzzy VC Combinatorics and Distality in Continuous Logic. arXiv preprint arXiv:2310.04393.

[24] Fox, J., Pach, J., Suk, A. (2021). Bounded VC-dimension implies the Schur-Erdős conjecture. Combinatorica, 41(6), 803-813.

[25] Johnson, H. R. (2021). Binary strings of finite VC dimension. arXiv preprint arXiv:2101.06490.

[26] Janzing, D. (2018). Merging joint distributions via causal model classes with low VC dimension. arXiv preprint arXiv:1804.03206.

[27] Hüllermeier, E., Fallah Tehrani, A. (2012, July). On the vc-dimension of the choquet integral. In International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (pp. 42-50). Berlin, Heidelberg: Springer Berlin Heidelberg.

[28] Mohri, M. (2018). Foundations of machine learning.

[29] Cucker, F., Zhou, D. X. (2007). Learning theory: an approximation theory viewpoint (Vol. 24). Cambridge University Press.

[30] Shalev-Shwartz, S., Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

[31] Truong, L. V. (2022). On rademacher complexity-based generalization bounds for deep learning. arXiv preprint arXiv:2208.04284.

[32] Gnecco, G., and Sanguineti, M. (2008). Approximation error bounds via Rademacher complexity. Applied Mathematical Sciences, 2, 153-176.

[33] Astashkin, S. V. (2010). Rademacher functions in symmetric spaces. Journal of Mathematical Sciences, 169(6), 725-886.

[34] Ying and Campbell (2010). Rademacher chaos complexities for learning the kernel problem. Neural computation, 22(11), 2858-2886.

[35] Zhu, J., Gibson, B., and Rogers, T. T. (2009). Human rademacher complexity. Advances in neural information processing systems, 22.

[36] Astashkin, S. V., Astashkin, S. V., and Mazlum. (2020). The Rademacher system in function spaces. Basel: Birkhäuser.

[37] Sachs, S., van Erven, T., Hodgkinson, L., Khanna, R., and Şimşekli, U. (2023, July). Generalization Guarantees via Algorithm-dependent Rademacher Complexity. In The Thirty Sixth Annual Conference on Learning Theory (pp. 4863-4880). PMLR.

[38] Ma and Wang (2020). Rademacher complexity and the generalization error of residual networks. Communications in Mathematical Sciences, 18(6), 1755-1774.

[39] Bartlett, P. L., and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research, 3(Nov), 463-482.

[40] Bartlett, P. L., and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research, 3(Nov), 463-482.

[41] McDonald, D. J., and Shalizi, C. R. (2011). Rademacher complexity of stationary sequences. arXiv preprint arXiv:1106.0730.

[42] Abderachid, S., and Kenza, B. EMBEDDINGS IN RIEMANN–LIOUVILLE FRACTIONAL SOBOLEV SPACES AND APPLICATIONS.

[43] Giang, T. H., Tri, N. M., and Tuan, D. A. (2024). On some Sobolev and Pólya-Sezgö type inequalities with weights and applications. arXiv preprint arXiv:2412.15490.

[44] Ruiz, P. A., and Fragkiadaki, V. (2024). Fractional Sobolev embeddings and algebra property: A dyadic view. arXiv preprint arXiv:2412.12051.

[45] Bilalov, B., Mamedov, E., Sezer, Y., and Nasibova, N. (2025). Compactness in Banach function spaces: Poincaré and Friedrichs inequalities. Rendiconti del Circolo Matematico di Palermo Series 2, 74(1), 68.

[46] Cheng, M., and Shao, K. (2025). Ground states of the inhomogeneous nonlinear fractional Schrödinger-Poisson equations. Complex Variables and Elliptic Equations, 1-17.

[47] Wei, J., and Zhang, L. (2025). Ground State Solutions of Nehari-Pohozaev Type for Schrödinger-Poisson Equation with Zero-Mass and Weighted Hardy Sobolev Subcritical Exponent. The Journal of Geometric Analysis, 35(2), 48.

[48] Zhang, X., and Qi, W. (2025). Multiplicity result on a class of nonhomogeneous quasilinear elliptic system with small perturbations in $\mathbb{R}^N$. arXiv preprint arXiv:2501.01602.

[49] Xiao, J., and Yue, C. (2025). A Trace Principle for Fractional Laplacian with an Application to Image Processing. La Matematica, 1-26.

[50] Pesce, A., and Portaro, S. (2025). Fractional Sobolev spaces related to an ultraparabolic operator. arXiv preprint arXiv:2501.05898.

[51] LASSOUED, D. (2026). A STUDY OF FUNCTIONS ON THE TORUS AND MULTI-PERIODIC FUNCTIONS. Kragujevac Journal of Mathematics, 50(2), 297-337.

[52] Chen, H., Chen, H. G., and Li, J. N. (2024). Sharp embedding results and geometric inequalities for Hö rmander vector fields. arXiv preprint arXiv:2404.19393.

[53] Adams, R. A., and Fournier, J. J. (2003). Sobolev spaces. Elsevier.

[54] Brezis, H., and Brézis, H. (2011). Functional analysis, Sobolev spaces and partial differential equations (Vol. 2, No. 3, p. 5). New York: Springer.

[55] Evans, L. C. (2022). Partial differential equations (Vol. 19). American Mathematical Society.

[56] Maz'â, V. G. (2011). Sobolev Spaces: With Applications to Elliptic Partial Differential Equations. Springer.

[57] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[58] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4), 303-314.

[59] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information theory, 39(3), 930-945.

[60] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. Acta numerica, 8, 143-195.

[61] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. Advances in neural information processing systems, 30.

[62] Hanin, B., and Sellke, M. (2017). Approximating continuous functions by relu nets of minimal width. arXiv preprint arXiv:1710.11278.

[63] Garcıa-Cervera, C. J., Kessler, M., Pedregal, P., and Periago, F. Universal approximation of set-valued maps and DeepONet approximation of the controllability map.

[64] Majee, S., Abhishek, A., Strauss, T., and Khan, T. (2024). MCMC-Net: Accelerating Markov Chain Monte Carlo with Neural Networks for Inverse Problems. arXiv preprint arXiv:2412.16883.

[65] Toscano, J. D., Wang, L. L., and Karniadakis, G. E. (2024). KKANs: Kurkova-Kolmogorov-Arnold Networks and Their Learning Dynamics. arXiv preprint arXiv:2412.16738.

[66] Son, H. (2025). ELM-DeepONets: Backpropagation-Free Training of Deep Operator Networks via Extreme Learning Machines. arXiv preprint arXiv:2501.09395.

[67] Rudin, W. (1964). Principles of mathematical analysis (Vol. 3). New York: McGraw-hill.

[68] Stein, E. M., and Shakarchi, R. (2009). Real analysis: measure theory, integration, and Hilbert spaces. Princeton University Press.

[69] Conway, J. B. (2019). A course in functional analysis (Vol. 96). Springer.

[70] Dieudonné, J. (2020). History of Functional Analyais. In Functional Analysis, Holomorphy, and Approximation Theory (pp. 119-129). CRC Press.

[71] Folland, G. B. (1999). Real analysis: modern techniques and their applications (Vol. 40). John Wiley and Sons.

[72] Sugiura, S. (2024). On the Universality of Reservoir Computing for Uniform Approximation.

[73] LIU, Y., LIU, S., HUANG, Z., and ZHOU, P. NORMED MODULES AND THE CATEGORIFICATION OF INTEGRATIONS, SERIES EXPANSIONS, AND DIFFERENTIATIONS.

[74] Barreto, D. M. (2025). Stone-Weierstrass Theorem.

[75] Chang, S. Y., and Wei, Y. (2024). Generalized Choi–Davis–Jensen's Operator Inequalities and Their Applications. Symmetry, 16(9), 1176.

[76] Caballer, M., Dantas, S., and Rodríguez-Vidanes, D. L. (2024). Searching for linear structures in the failure of the Stone-Weierstrass theorem. arXiv preprint arXiv:2405.06453.

[77] Chen, D. (2024). The Machado–Bishop theorem in the uniform topology. Journal of Approximation Theory, 304, 106085.

[78] Rafiei, H., and Akbarzadeh-T, M. R. (2024). Hedge-embedded Linguistic Fuzzy Neural Networks for Systems Identification and Control. IEEE Transactions on Artificial Intelligence.

[79] Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In Doklady Akademii Nauk (Vol. 114, No. 5, pp. 953-956). Russian Academy of Sciences.

[80] Arnold, V. I. (2009). On the representation of functions of several variables as a superposition of functions of a smaller number of variables. Collected works: Representations of functions, celestial mechanics and KAM theory, 1957–1965, 25-46.

[81] Lorentz, G. G. (1966). Approximation of functions, athena series. Selected Topics in Mathematics.

[82] Guilhoto, L. F., and Perdikaris, P. (2024). Deep learning alternatives of the Kolmogorov superposition theorem. arXiv preprint arXiv:2410.01990.

[83] Alhafiz, M. R., Zakaria, K., Dung, D. V., Palar, P. S., Dwianto, Y. B., and Zuhal, L. R. (2025). Kolmogorov-Arnold Networks for Data-Driven Turbulence Modeling. In AIAA SCITECH 2025 Forum (p. 2047).

[84] Lorencin, I., Mrzljak, V., Poljak, I., and Etinger, D. (2024, September). Prediction of CODLAG Propulsion System Parameters Using Kolmogorov-Arnold Network. In 2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY) (pp. 173-178). IEEE.

[85] Trevisan, D., Cassara, P., Agazzi, A., and Scardera, S. NTK Analysis of Knowledge Distillation.

[86] Bonfanti, A., Bruno, G., and Cipriani, C. (2024). The Challenges of the Nonlinear Regime for Physics-Informed Neural Networks. arXiv preprint arXiv:2402.03864.

[87] Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. Advances in neural information processing systems, 31.

[88] Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. Advances in neural information processing systems, 32.

[89] Yang, G., and Hu, E. J. (2020). Feature learning in infinite-width neural networks. arXiv preprint arXiv:2011.14522.

[90] Xiang, L., Dudziak, Ł., Abdelfattah, M. S., Chau, T., Lane, N. D., and Wen, H. (2021). Zero-Cost Operation Scoring in Differentiable Architecture Search. arXiv preprint arXiv:2106.06799.

[91] Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. Advances in neural information processing systems, 32.

[92] McAllester, D. A. (1999, July). PAC-Bayesian model averaging. In Proceedings of the twelfth annual conference on Computational learning theory (pp. 164-170).

[93] Catoni, O. (2007). PAC-Bayesian supervised classification: the thermodynamics of statistical learning. arXiv preprint arXiv:0712.0248.

[94] Germain, P., Lacasse, A., Laviolette, F., and Marchand, M. (2009, June). PAC-Bayesian learning of linear classifiers. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 353-360).

[95] Seeger, M. (2002). PAC-Bayesian generalisation error bounds for Gaussian process classification. Journal of machine learning research, 3(Oct), 233-269.

[96] Alquier, P., Ridgway, J., and Chopin, N. (2016). On the properties of variational approximations of Gibbs posteriors. Journal of Machine Learning Research, 17(236), 1-41.

[97] Dziugaite, G. K., and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. arXiv preprint arXiv:1703.11008.

[98] Rivasplata, O., Kuzborskij, I., Szepesvári, C., and Shawe-Taylor, J. (2020). PAC-Bayes analysis beyond the usual bounds. Advances in Neural Information Processing Systems, 33, 16833-16845.

[99] Lever, G., Laviolette, F., and Shawe-Taylor, J. (2013). Tighter PAC-Bayes bounds through distribution-dependent priors. Theoretical Computer Science, 473, 4-28.

[100] Rivasplata, O., Parrado-Hernández, E., Shawe-Taylor, J. S., Sun, S., and Szepesvári, C. (2018). PAC-Bayes bounds for stable algorithms with instance-dependent priors. Advances in Neural Information Processing Systems, 31.

[101] Lindemann, L., Zhao, Y., Yu, X., Pappas, G. J., and Deshmukh, J. V. (2024). Formal verification and control with conformal prediction. arXiv preprint arXiv:2409.00536.

[102] Jin, G., Wu, S., Liu, J., Huang, T., and Mu, R. (2025). Enhancing Robust Fairness via Confusional Spectral Regularization. arXiv preprint arXiv:2501.13273.

[103] Ye, F., Xiao, J., Ma, W., Jin, S., and Yang, Y. (2025). Detecting small clusters in the stochastic block model. Statistical Papers, 66(2), 37.

[104] Bhattacharjee, A., and Bharadwaj, P. (2025). Coherent Spectral Feature Extraction Using Symmetric Autoencoders. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[105] Wu, Q., Hu, B., Liu, C. et al. (2025). Velocity Analysis Using High-resolution Hyperbolic Radon Transform with $L_{q_1} - L_{q_2}$ Regularization. Pure Appl. Geophys.

[106] Ortega, I., Hannigan, J. W., Baier, B. C., McKain, K., and Smale, D. (2025). Advancing CH 4 and N 2 O retrieval strategies for NDACC/IRWG high-resolution direct-sun FTIR Observations. EGUsphere, 2025, 1-32.

[107] Kazmi, S. H. A., Hassan, R., Qamar, F., Nisar, K., and Al-Betar, M. A. (2025). Federated Conditional Variational Auto Encoders for Cyber Threat Intelligence: Tackling Non-IID Data in SDN Environments. IEEE Access.

[108] Zhao, Y., Bi, Z., Zhu, P., Yuan, A., and Li, X. (2025). Deep Spectral Clustering with Projected Adaptive Feature Selection. IEEE Transactions on Geoscience and Remote Sensing.

[109] Saranya, S., and Menaka, R. (2025). A Quantum-Based Machine Learning Approach for Autism Detection using Common Spatial Patterns of EEG Signals. IEEE Access.

[110] Dhalbisoi, S., Mohapatra, A., and Rout, A. (2024, March). Design of Cell-Free Massive MIMO for Beyond 5G Systems with MMSE and RZF Processing. In International Conference on Machine Learning, IoT and Big Data (pp. 263-273). Singapore: Springer Nature Singapore.

[111] Wei, C., Li, Z., Hu, T., Zhao, M., Sun, Z., Jia, K., ... and Jiang, S. (2025). Model-based convolution neural network for 3D Near-infrared spectral tomography. IEEE Transactions on Medical Imaging.

[112] Goodfellow, I. (2016). Deep learning (Vol. 196). MIT press.

[113] Haykin, S. (2009). Neural networks and learning machines, 3/E. Pearson Education India.

[114] Schmidhuber, J. (2015). Deep learning in neural networks: An overview.

[115] Bishop, C. M., and Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.

[116] Poggio, T., and Smale, S. (2003). The mathematics of learning: Dealing with data. Notices of the AMS, 50(5), 537-544.

[117] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[118] Tishby, N., and Zaslavsky, N. (2015, April). Deep learning and the information bottleneck principle. In 2015 ieee information theory workshop (itw) (pp. 1-5). IEEE.

[119] Sorrenson, P. (2025). Free-Form Flows: Generative Models for Scientific Applications (Doctoral dissertation).

[120] Liu, W., and Shi, X. (2025). An Enhanced Neural Network Forecasting System for the July Precipitation over the Middle-Lower Reaches of the Yangtze River.

[121] Das, P., Mondal, D., Islam, M. A., Al Mohotadi, M. A., and Roy, P. C. (2025). Analytical Finite-Integral-Transform and Gradient-Enhanced Machine Learning Approach for Thermoelastic Analysis of FGM Spherical Structures with Arbitrary Properties. Theoretical and Applied Mechanics Letters, 100576.

[122] Zhang, R. (2025). Physics-informed Parallel Neural Networks for the Identification of Continuous Structural Systems.

[123] Ali, S., and Hussain, A. (2025). A neuro-intelligent heuristic approach for performance prediction of triangular fuzzy flow system. Proceedings of the Institution of Mechanical Engineers, Part N: Journal of Nanomaterials, Nanoengineering and Nanosystems, 23977914241310569.

[124] Li, S. (2025). Scalable, generalizable, and offline methods for imperfect-information extensive-form games.

[125] Hu, T., Jin, B., and Wang, F. (2025). An Iterative Deep Ritz Method for Monotone Elliptic Problems. Journal of Computational Physics, 113791.

[126] Chen, P., Zhang, A., Zhang, S., Dong, T., Zeng, X., Chen, S., ... and Zhou, Q. (2025). Maritime near-miss prediction framework and model interpretation analysis method based on Transformer neural network model with multi-task classification variables. Reliability Engineering and System Safety, 110845.

[127] Sun, G., Liu, Z., Gan, L., Su, H., Li, T., Zhao, W., and Sun, B. (2025). SpikeNAS-Bench: Benchmarking NAS Algorithms for Spiking Neural Network Architecture. IEEE Transactions on Artificial Intelligence.

[128] Zhang, Z., Wang, X., Shen, J., Zhang, M., Yang, S., Zhao, W., ... and Wang, J. (2025). Unfixed Bias Iterator: A New Iterative Format. IEEE Access.

[129] Rosa, G. J. (2010). The Elements of Statistical Learning: Data Mining, Inference, and Prediction by HASTIE, T., TIBSHIRANI, R., and FRIEDMAN, J.

[130] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.

[131] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[132] Zou, H., and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society Series B: Statistical Methodology, 67(2), 301-320.

[133] Vapnik, V. (2013). The nature of statistical learning theory. Springer science and business media.

[134] Ng, A. Y. (2004, July). Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In Proceedings of the twenty-first international conference on Machine learning (p. 78).

[135] Li, T. (2025). Optimization of Clinical Trial Strategies for Anti-HER2 Drugs Based on Bayesian Optimization and Deep Learning.

[136] Yasuda, M., and Sekimoto, K. (2024). Gaussian-discrete restricted Boltzmann machine with sparse-regularized hidden layer. Behaviormetrika, 1-19.

[137] Xiaodong Luo, William C. Cruz, Xin-Lei Zhang, Heng Xiao, (2023), Hyper-parameter optimization for improving the performance of localization in an iterative ensemble smoother, Geoenergy Science and Engineering, Volume 231, Part B, 212404

[138] Alrayes, F.S., Maray, M., Alshuhail, A. et al. (2025) Privacy-preserving approach for IoT networks using statistical learning with optimization algorithm on high-dimensional big data environment. Sci Rep 15, 3338. https://doi.org/10.1038/s41598-025-87454-1

[139] Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., and Rhee, W. (2020). Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks. IEEE access, 8, 52588-52608.

[140] IBRAHIM, M. M. W. (2025). Optimizing Tuberculosis Treatment Predictions: A Comparative Study of XGBoost with Hyperparameter in Penang, Malaysia. Sains Malaysiana, 54(1), 3741-3752.

[141] Abdel-salam, M., Elhoseny, M. and El-hasnony, I.M. Intelligent and Secure Evolved Framework for Vaccine Supply Chain Management Using Machine Learning and Blockchain. SN COMPUT. SCI. 6, 121 (2025). https://doi.org/10.1007/s42979-024-03609-3

[142] Vali, M. H. (2025). Vector quantization in deep neural networks for speech and image processing.

[143] Vincent, A.M., Jidesh, P. An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. Sci Rep 13, 4737 (2023). https://doi.org/10.1038/s41598-023-32027-3

[144] Razavi-Termeh, S. V., Sadeghi-Niaraki, A., Ali, F., and Choi, S. M. (2025). Improving flood-prone areas mapping using geospatial artificial intelligence (GeoAI): A non-parametric algorithm enhanced by math-based metaheuristic algorithms. Journal of Environmental Management, 375, 124238.

[145] Kiran, M., and Ozyildirim, M. (2022). Hyperparameter tuning for deep reinforcement learning applications. arXiv preprint arXiv:2201.11182.

[146] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[147] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

[148] Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[149] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[150] Cohen, T., and Welling, M. (2016, June). Group equivariant convolutional networks. In International conference on machine learning (pp. 2990-2999). PMLR.

[151] Zeiler, M. D., and Fergus, R. (2014). Visualizing and understanding convolutional networks. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13 (pp. 818-833). Springer International Publishing.

[152] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022).

[153] Lin, M. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[154] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[155] Bensaid, B., Poëtte, G., and Turpault, R. (2024). Convergence of the Iterates for Momentum and RMSProp for Local Smooth Functions: Adaptation is the Key. arXiv preprint arXiv:2407.15471.

[156] Liu, Q., and Ma, W. (2024). The Epochal Sawtooth Effect: Unveiling Training Loss Oscillations in Adam and Other Optimizers. arXiv preprint arXiv:2410.10056.

[157] Li, H. (2024). Smoothness and Adaptivity in Nonlinear Optimization for Machine Learning Applications (Doctoral dissertation, Massachusetts Institute of Technology).

[158] Heredia, C. (2024). Modeling AdaGrad, RMSProp, and Adam with Integro-Differential Equations. arXiv preprint arXiv:2411.09734.

[159] Ye, Q. (2024). Preconditioning for Accelerated Gradient Descent Optimization and Regularization. arXiv preprint arXiv:2410.00232.

[160] Compagnoni, E. M., Liu, T., Islamov, R., Proske, F. N., Orvieto, A., and Lucchi, A. (2024). Adaptive Methods through the Lens of SDEs: Theoretical Insights on the Role of Noise. arXiv preprint arXiv:2411.15958.

[161] Yao, B., Zhang, Q., Feng, R., and Wang, X. (2024). System response curve based first-order optimization algorithms for cyber-physical-social intelligence. Concurrency and Computation: Practice and Experience, 36(21), e8197.

[162] Wen, X., and Lei, Y. (2024, June). A Fast ADMM Framework for Training Deep Neural Networks Without Gradients. In 2024 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[163] Hannibal, S., Jentzen, A., and Thang, D. M. (2024). Non-convergence to global minimizers in data driven supervised deep learning: Adam and stochastic gradient descent optimization provably fail to converge to global minimizers in the training of deep neural networks with ReLU activation. arXiv preprint arXiv:2410.10533.

[164] Yang, Z. (2025). Adaptive Biased Stochastic Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[165] Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[166] Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237.

[167] Jin, L., Nong, H., Chen, L., and Su, Z. (2024). A Method for Enhancing Generalization of Adam by Multiple Integrations. arXiv preprint arXiv:2412.12473.

[168] Adly, A. M. (2024). EXAdam: The Power of Adaptive Cross-Moments. arXiv preprint arXiv:2412.20302.

[169] Liu, Y., Cao, Y., and Lin, J. Convergence Analysis of the ADAM Algorithm for Linear Inverse Problems.

[170] Yang, Z. (2025). Adaptive Biased Stochastic Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[171] Park, K., and Lee, S. (2024). SMMF: Square-Matricized Momentum Factorization for Memory-Efficient Optimization. arXiv preprint arXiv:2412.08894.

[172] Mahjoubi, M. A., Lamrani, D., Saleh, S., Moutaouakil, W., Ouhmida, A., Hamida, S., ... and Raihani, A. (2025). Optimizing ResNet50 Performance Using Stochastic Gradient Descent on MRI Images for Alzheimer's Disease Classification. Intelligence-Based Medicine, 100219.

[173] Seini, A. B., and Adam, I. O. (2024). HUMAN-AI COLLABORATION FOR ADAPTIVE WORKING AND LEARNING OUTCOMES: AN ACTIVITY THEORY PERSPECTIVE.

[174] Teessar, J. (2024). The Complexities of Truthful Responding in Questionnaire-Based Research: A Comprehensive Analysis.

[175] Lauand, C. K., and Meyn, S. (2025). Markovian Foundations for Quasi-Stochastic Approximation. SIAM Journal on Control and Optimization, 63(1), 402-430.

[176] Maranjyan, A., Tyurin, A., and Richtárik, P. (2025). Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity. arXiv preprint arXiv:2501.16168.

[177] Gao, Z., and Gündüz, D. (2025). Graph Neural Networks over the Air for Decentralized Tasks in Wireless Networks. IEEE Transactions on Signal Processing.

[178] Yoon, T., Choudhury, S., and Loizou, N. (2025). Multiplayer Federated Learning: Reaching Equilibrium with Less Communication. arXiv preprint arXiv:2501.08263.

[179] Verma, K., and Maiti, A. (2025). Sine and cosine based learning rate for gradient descent method. Applied Intelligence, 55(5), 352.

[180] Borowski, M., and Miasojedow, B. (2025). Convergence of projected stochastic approximation algorithm. arXiv e-prints, arXiv-2501.

[181] Dong, K., Chen, S., Dan, Y., Zhang, L., Li, X., Liang, W., ... and Sun, Y. (2025). A new perspective on brain stimulation interventions: Optimal stochastic tracking control of brain network dynamics. arXiv preprint arXiv:2501.08567.

[182] Jiang, Y., Kang, H., Liu, J., and Xu, D. (2025). On the Convergence of Decentralized Stochastic Gradient Descent with Biased Gradients. IEEE Transactions on Signal Processing.

[183] Sonobe, N., Momozaki, T., and Nakagawa, T. (2025). Sampling from Density power divergence-based Generalized posterior distribution via Stochastic optimization. arXiv preprint arXiv:2501.07790.

[184] Zhang, X., and Jia, G. (2025). Convergence of Policy Gradient for Stochastic Linear Quadratic Optimal Control Problems in Infinite Horizon. Journal of Mathematical Analysis and Applications, 129264.

[185] Thiriveedhi, A., Ghanta, S., Biswas, S., and Pradhan, A. K. (2025). ALL-Net: integrating CNN and explainable-AI for enhanced diagnosis and interpretation of acute lymphoblastic leukemia. PeerJ Computer Science, 11, e2600.

[186] Ramos-Briceño, D. A., Flammia-D'Aleo, A., Fernández-López, G., Carrión-Nessi, F. S., and Forero-Peña, D. A. (2025). Deep learning-based malaria parasite detection: convolutional neural networks model for accurate species identification of Plasmodium falciparum and Plasmodium vivax. Scientific Reports, 15(1), 3746.

[187] Espino-Salinas, C. H., Luna-García, H., Cepeda-Argüelles, A., Trejo-Vázquez, K., Flores-Chaires, L. A., Mercado Reyna, J., ... and Villalba-Condori, K. O. (2025). Convolutional Neural Network for Depression and Schizophrenia Detection. Diagnostics, 15(3), 319.

[188] Ran, T., Huang, W., Qin, X., Xie, X., Deng, Y., Pan, Y., ... and Zou, D. (2025). Liquid-based cytological diagnosis of pancreatic neuroendocrine tumors using hyperspectral imaging and deep learning. EngMedicine, 2(1), 100059.

[189] Araujo, B. V. S., Rodrigues, G. A., de Oliveira, J. H. P., Xavier, G. V. R., Lebre, U., Cordeiro, C., ... and Ferreira, T. V. (2025). Monitoring ZnO surge arresters using convolutional neural networks and image processing techniques combined with signal alignment. Measurement, 116889.

[190] Sari, I. P., Elvitaria, L., and Rudiansyah, R. (2025). Data-driven approach for batik pattern classification using convolutional neural network (CNN). Jurnal Mandiri IT, 13(3), 323-331.

[191] Wang, D., An, K., Mo, Y., Zhang, H., Guo, W., and Wang, B. Cf-Wiad: Consistency Fusion with Weighted Instance and Adaptive Distribution for Enhanced Semi-Supervised Skin Lesion Classification. Available at SSRN 5109182.

[192] Cai, P., Zhang, Y., He, H., Lei, Z., and Gao, S. (2025). DFNet: A Differential Feature-Incorporated Residual Network for Image Recognition. Journal of Bionic Engineering, 1-14.

[193] Vishwakarma, A. K., and Deshmukh, M. (2025). CNNM-FDI: Novel Convolutional Neural Network Model for Fire Detection in Images. IETE Journal of Research, 1-14.

[194] Ranjan, P., Kaushal, A., Girdhar, A., and Kumar, R. (2025). Revolutionizing hyperspectral image classification for limited labeled data: unifying autoencoder-enhanced GANs with convolutional neural networks and zero-shot learning. Earth Science Informatics, 18(2), 1-26.

[195] Naseer, A., and Jalal, A. Multimodal Deep Learning Framework for Enhanced Semantic Scene Classification Using RGB-D Images.

[196] Wang, Z., and Wang, J. (2025). Personalized Icon Design Model Based on Improved Faster-RCNN. Systems and Soft Computing, 200193.

[197] Ramana, R., Vasudevan, V., and Murugan, B. S. (2025). Spectral Pyramid Pooling and Fused Keypoint Generation in ResNet-50 for Robust 3D Object Detection. IETE Journal of Research, 1-13.

[198] Shin, S., Land, O., Seider, W., Lee, J., and Lee, D. (2025). Artificial Intelligence-Empowered Automated Double Emulsion Droplet Library Generation.

[199] Taca, B. S., Lau, D., and Rieder, R. (2025). A comparative study between deep learning approaches for aphid classification. IEEE Latin America Transactions, 23(3), 198-204.

[200] Ulaş, B., Szklenár, T., and Szabó, R. (2025). Detection of Oscillation-like Patterns in Eclipsing Binary Light Curves using Neural Network-based Object Detection Algorithms. arXiv preprint arXiv:2501.17538.

[201] Valensi, D., Lupu, L., Adam, D., and Topilsky, Y. Semi-Supervised Learning, Foundation Models and Image Processing for Pleural Line Detection and Segmentation in Lung Ultrasound. Foundation Models and Image Processing for Pleural Line Detection and Segmentation in Lung Ultrasound.

[202] V, A., V, P. and Kumar, D. An effective object detection via BS2ResNet and LTK-Bi-LSTM. Multimed Tools Appl (2025). https://doi.org/10.1007/s11042-024-20433-2

[203] Zhu, X., Chen, W., and Jiang, Q. (2025). High-transferability black-box attack of binary image segmentation via adversarial example augmentation. Displays, 102957.

[204] Guo, X., Zhu, Y., Li, S., Wu, S., and Liu, S. (2025). Research and Implementation of Agronomic Entity and Attribute Extraction Based on Target Localization. Agronomy, 15(2), 354.

[205] Yousif, M., Jassam, N. M., Salim, A., Bardan, H. A., Mutlak, A. F., Sallibi, A. D., and Ataalla, A. F. Melanoma Skin Cancer Detection Using Deep Learning Methods and Binary GWO Algorithm.

[206] Rahman, S. I. U., Abbas, N., Ali, S., Salman, M., Alkhayat, A., Khan, J., ... and Gu, Y. H. (2025). Deep Learning and Artificial Intelligence-Driven Advanced Methods for Acute Lymphoblastic Leukemia Identification and Classification: A Systematic Review. Comput Model Eng Sci, 142(2).

[207] Pratap Joshi, K., Gowda, V. B., Bidare Divakarachari, P., Siddappa Parameshwarappa, P., and Patra, R. K. (2025). VSA-GCNN: Attention Guided Graph Neural Networks for Brain Tumor Segmentation and Classification. Big Data and Cognitive Computing, 9(2), 29.

[208] Ng, B., Eyre, K., and Chetrit, M. (2025). Prediction of ischemic cardiomyopathy using a deep neural network with non-contrast cine cardiac magnetic resonance images. Journal of Cardiovascular Magnetic Resonance, 27.

[209] Nguyen, H. T., Lam, T. B., Truong, T. T. N., Duong, T. D., and Dinh, V. Q. Mv-Trams: An Efficient Tumor Region-Adapted Mammography Synthesis Under Multi-View Diagnosis. Available at SSRN 5109180.

[210] Chen, W., Xu, T., and Zhou, W. (2025). Task-based Regularization in Penalized Least-Squares for Binary Signal Detection Tasks in Medical Image Denoising. arXiv preprint arXiv:2501.18418.

[211] Pradhan, P. D., Talmale, G., and Wazalwar, S. Deep dive into precision (DDiP): Unleashing advanced deep learning approaches in diabetic retinopathy research for enhanced detection and classification of retinal abnormalities. In Recent Advances in Sciences, Engineering, Information Technology and Management (pp. 518-530). CRC Press.

[212] Örenç, S., Acar, E., Özerdem, M. S., Şahin, S., and Kaya, A. (2025). Automatic Identification of Adenoid Hypertrophy via Ensemble Deep Learning Models Employing X-ray Adenoid Images. Journal of Imaging Informatics in Medicine, 1-15.

[213] Jiang, M., Wang, S., Chan, K. H., Sun, Y., Xu, Y., Zhang, Z., ... and Tan, T. (2025). Multimodal Cross Global Learnable Attention Network for MR images denoising with arbitrary modal missing. Computerized Medical Imaging and Graphics, 102497.

[214] Al-Haidri, W., Levchuk, A., Zotov, N., Belousova, K., Ryzhkov, A., Fokin, V., ... and Brui, E. (2025). Quantitative analysis of myocardial fibrosis using a deep learning-based framework applied to the 17-Segment model. Biomedical Signal Processing and Control, 105, 107555.

[215] Osorio, S. L. J., Ruiz, M. A. R., Mendez-Vazquez, A., and Rodriguez-Tello, E. (2024). Fourier Series Guided Design of Quantum Convolutional Neural Networks for Enhanced Time Series Forecasting. arXiv preprint arXiv:2404.15377.

[216] Umeano, C., and Kyriienko, O. (2024). Ground state-based quantum feature maps. arXiv preprint arXiv:2404.07174.

[217] Liu, N., He, X., Laurent, T., Di Giovanni, F., Bronstein, M. M., and Bresson, X. (2024). Advancing Graph Convolutional Networks via General Spectral Wavelets. arXiv preprint arXiv:2405.13806.

[218] Vlasic, A. (2024). Quantum Circuits, Feature Maps, and Expanded Pseudo-Entropy: A Categorical Theoretic Analysis of Encoding Real-World Data into a Quantum Computer. arXiv preprint arXiv:2410.22084.

[219] Kim, M., Hioka, Y., and Witbrock, M. (2024). Neural Fourier Modelling: A Highly Compact Approach to Time-Series Analysis. arXiv preprint arXiv:2410.04703.

[220] Xie, Y., Daigavane, A., Kotak, M., and Smidt, T. (2024). The price of freedom: Exploring tradeoffs between expressivity and computational efficiency in equivariant tensor products. In ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling.

[221] Liu, G., Wei, Z., Zhang, H., Wang, R., Yuan, A., Liu, C., ... and Cao, G. (2024, April). Extending Implicit Neural Representations for Text-to-Image Generation. In ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3650-3654). IEEE.

[222] Zhang, M. (2024). Lock-in spectrum: a tool for representing long-term evolution of bearing fault in the time–frequency domain using vibration signal. Sensor Review, 44(5), 598-610.

[223] Hamed, M., and Lachiri, Z. (2024, July). Expressivity Transfer In Transformer-Based Text-To-Speech Synthesis. In 2024 IEEE 7th International Conference on Advanced Technologies, Signal and Image Processing (ATSIP) (Vol. 1, pp. 443-448). IEEE.

[224] Lehmann, F., Gatti, F., Bertin, M., Grenié, D., and Clouteau, D. (2024). Uncertainty propagation from crustal geologies to rock-site ground motion with a Fourier Neural Operator. European Journal of Environmental and Civil Engineering, 28(13), 3088-3105.

[225] Jurafsky, D. (2000). Speech and language processing.

[226] Manning, C., and Schutze, H. (1999). Foundations of statistical natural language processing. MIT press.

[227] Liu, Y., and Zhang, M. (2018). Neural network methods for natural language processing.

[228] Allen, J. (1988). Natural language understanding. Benjamin-Cummings Publishing Co., Inc..

[229] Li, Z., Zhao, Y., Zhang, X., Han, H., and Huang, C. (2025). Word embedding factor based multi-head attention. Artificial Intelligence Review, 58(4), 1-21.

[230] Hempelmann, C. F., Rayz, J., Dong, T., and Miller, T. (2025, January). Proceedings of the 1st Workshop on Computational Humor (CHum). In Proceedings of the 1st Workshop on Computational Humor (CHum).

[231] Koehn, P. (2009). Statistical machine translation. Cambridge University Press.

[232] Eisenstein, J. (2019). Introduction to natural language processing. The MIT Press.

[233] Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. IEEE transactions on neural networks and learning systems, 32(2), 604-624.

[234] Mitkov, R. (Ed.). (2022). The Oxford handbook of computational linguistics. Oxford university press.

[235] Liu, X., Tao, Z., Jiang, T., Chang, H., Ma, Y., and Huang, X. (2024). ToDA: Target-oriented Diffusion Attacker against Recommendation System. arXiv preprint arXiv:2401.12578.

[236] Çekik, R. (2025). Effective Text Classification Through Supervised Rough Set-Based Term Weighting. Symmetry, 17(1), 90.

[237] Zhu, H., Xia, J., Liu, R., and Deng, B. (2025). SPIRIT: Structural Entropy Guided Prefix Tuning for Hierarchical Text Classification. Entropy, 27(2), 128.

[238] Matrane, Y., Benabbou, F., and Ellaky, Z. (2024). Enhancing Moroccan Dialect Sentiment Analysis through Optimized Preprocessing and transfer learning Techniques. IEEE Access.

[239] Moqbel, M., and Jain, A. (2025). Mining the truth: A text mining approach to understanding perceived deceptive counterfeits and online ratings. Journal of Retailing and Consumer Services, 84, 104149.

[240] Kumar, V., Iqbal, M. I., and Rathore, R. (2025). Natural Language Processing (NLP) in Disease Detection—A Discussion of How NLP Techniques Can Be Used to Analyze and Classify Medical Text Data for Disease Diagnosis. AI in Disease Detection: Advancements and Applications, 53-75.

[241] Yin, S. (2024). The Current State and Challenges of Aspect-Based Sentiment Analysis. Applied and Computational Engineering, 114, 25-31.

[242] Raghavan, M. (2024). Are you who AI says you are? Exploring the role of Natural Language Processing algorithms for "predicting" personality traits from text (Doctoral dissertation, University of South Florida).

[243] Semeraro, A., Vilella, S., Improta, R., De Duro, E. S., Mohammad, S. M., Ruffo, G., and Stella, M. (2025). EmoAtlas: An emotional network analyzer of texts that merges psychological lexicons, artificial intelligence, and network science. Behavior Research Methods, 57(2), 77.

[244] Cai, F., and Liu, X. Data Analytics for Discourse Analysis with Python: The Case of Therapy Talk, by Dennis Tay. New York: Routledge, 2024. ISBN: 9781032419015 (HB: USD 41.24), xiii+ 182 pages. Natural Language Processing, 1-4.

[245] Wu, Yonghui. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

[246] Hettiarachchi, H., Ranasinghe, T., Rayson, P., Mitkov, R., Gaber, M., Premasiri, D., ... and Uyangodage, L. (2024). Overview of the First Workshop on Language Models for Low-Resource Languages (LoResLM 2025). arXiv preprint arXiv:2412.16365.

[247] Das, B. R., and Sahoo, R. (2024). Word Alignment in Statistical Machine Translation: Issues and Challenges. Nov Joun of Appl Sci Res, 1 (6), 01-03.

[248] Oluwatoki, T. G., Adetunmbi, O. A., and Boyinbode, O. K. A Transformer-Based Yoruba to English Machine Translation (TYEMT) System with Rouge Score.

[249] UÇKAN, T., and KURT, E. Word Embeddings in NLP. PIONEER AND INNOVATIVE STUDIES IN COMPUTER SCIENCES AND ENGINEERING, 58.

[250] Pastor, G. C., Monti, J., Mitkov, R., and Hidalgo-Ternero, C. M. (2024). Recent Advances in Multiword Units in Machine Translation and Translation Technology. Recent Advances in Multiword Units in Machine Translation and Translation Technology.

[251] Fernandes, R. M. Decoding spatial semantics: a comparative analysis of the performance of open-source LLMs against NMT systems in translating EN-PT-BR subtitles (Doctoral dissertation, Universidade de SÃ£ o Paulo).

[252] Jozić, K. (2024). Testing ChatGPT's Capabilities as an English-Croatian Machine Translation System in a Real-World Setting: eTranslation versus ChatGPT at the European Central Bank (Doctoral dissertation, University of Zagreb. Faculty of Humanities and Social Sciences. Department of English language and literature).

[253] Yang, M. (2025). Adaptive Recognition of English Translation Errors Based on Improved Machine Learning Methods. International Journal of High Speed Electronics and Systems, 2540236.

[254] Linnemann, G. A., and Reimann, L. E. (2024). Artificial Intelligence as a New Field of Activity for Applied Social Psychology–A Reasoning for Broadening the Scope.

[255] Merkel, S., and Schorr, S. OPP: APPLICATION FIELDS and INNOVATIVE TECHNOLOGIES.

[256] Kushwaha, N. S., and Singh, P. (2022). Artificial Intelligence based Chatbot: A Case Study. Journal of Management and Service Science (JMSS), 2(1), 1-13.

[257] Macedo, P., Madeira, R. N., Santos, P. A., Mota, P., Alves, B., and Pereira, C. M. (2024). A Conversational Agent for Empowering People with Parkinson's Disease in Exercising Through Motivation and Support. Applied Sciences, 15(1), 223.

[258] Gupta, R., Nair, K., Mishra, M., Ibrahim, B., and Bhardwaj, S. (2024). Adoption and impacts of generative artificial intelligence: Theoretical underpinnings and research agenda. International Journal of Information Management Data Insights, 4(1), 100232.

[259] Foroughi, B., Iranmanesh, M., Yadegaridehkordi, E., Wen, J., Ghobakhloo, M., Senali, M. G., and Annamalai, N. (2025). Factors Affecting the Use of ChatGPT for Obtaining Shopping Information. International Journal of Consumer Studies, 49(1), e70008.

[260] Jandhyala, V. S. V. (2024). BUILDING AI CHATBOTS AND VIRTUAL ASSISTANTS: A TECHNICAL GUIDE FOR ASPIRING PROFESSIONALS. INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND INFORMATION TECHNOLOGY (IJRCAIT), 7(2), 448-463.

[261] Pavlović, N., and Savić, M. (2024). The Impact of the ChatGPT Platform on Consumer Experience in Digital Marketing and User Satisfaction. Theoretical and Practical Research in Economic Fields, 15(3), 636-646.

[262] Mannava, V., Mitrevski, A., and Plöger, P. G. (2024, August). Exploring the Suitability of Conversational AI for Child-Robot Interaction. In 2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN) (pp. 1821-1827). IEEE.

[263] Sherstinova, T., Mikhaylovskiy, N., Kolpashchikova, E., and Kruglikova, V. (2024, April). Bridging Gaps in Russian Language Processing: AI and Everyday Conversations. In 2024 35th Conference of Open Innovations Association (FRUCT) (pp. 665-674). IEEE.

[264] Lipton, Z. C. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv Preprint, CoRR, abs/1506.00019.

[265] Pascanu, R. (2013). On the difficulty of training recurrent neural networks. arXiv preprint arXiv:1211.5063.

[266] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148(34), 13.

[267] Hochreiter, S. (1997). Long Short-term Memory. Neural Computation MIT-Press.

[268] Kawakami, K. (2008). Supervised sequence labelling with recurrent neural networks (Doctoral dissertation, Ph. D. thesis).

[269] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[270] Bhattamishra, S., Patel, A., and Goyal, N. (2020). On the computational power of transformers and its implications in sequence modeling. arXiv preprint arXiv:2006.09286.

[271] Siegelmann, H. T. (1993). Theoretical foundations of recurrent neural networks.

[272] Sutton, R. S. (2018). Reinforcement learning: An introduction. A Bradford Book.

[273] Barto, A. G. (2021). Reinforcement Learning: An Introduction. By Richard's Sutton. SIAM Rev, 6(2), 423.

[274] Bertsekas, D. P. (1996). Neuro-dynamic programming. Athena Scientific.

[275] Kakade, S. M. (2003). On the sample complexity of reinforcement learning. University of London, University College London (United Kingdom).

[276] Szepesvári, C. (2022). Algorithms for reinforcement learning. Springer nature.

[277] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). PMLR.

[278] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... and Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[279] Konda, V., and Tsitsiklis, J. (1999). Actor-critic algorithms. Advances in neural information processing systems, 12.

[280] Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909.

[281] Mannor, S., Mansour, Y., and Tamar, A. (2022). Reinforcement Learning: Foundations. Online manuscript.

[282] Borkar, V. S., and Borkar, V. S. (2008). Stochastic approximation: a dynamical systems viewpoint (Vol. 9). Cambridge: Cambridge University Press.

[283] Takhsha, Amir Reza, Maryam Rastgarpour, and Mozhgan Naderi. "A Feature-Level Ensemble Model for COVID-19 Identification in CXR Images using Choquet Integral and Differential Evolution Optimization." arXiv preprint arXiv:2501.08241 (2025).

[284] Singh, P., and Raman, B. (2025). Graph Neural Networks: Extending Deep Learning to Graphs. In Deep Learning Through the Prism of Tensors (pp. 423-482). Singapore: Springer Nature Singapore.

[285] Yao, L., Shi, Q., Yang, Z., Shao, S., and Hariri, S. (2024). Development of an Edge Resilient ML Ensemble to Tolerate ICS Adversarial Attacks. arXiv preprint arXiv:2409.18244.

[286] Chen, K., Bi, Z., Niu, Q., Liu, J., Peng, B., Zhang, S., ... and Feng, P. (2024). Deep learning and machine learning, advancing big data analytics and management: Tensorflow pretrained models. arXiv preprint arXiv:2409.13566.

[287] Dumić, E. (2024). Learning neural network design with TensorFlow and Keras. In ICERI2024 Proceedings (pp. 10689-10696). IATED.

[288] Bajaj, K., Bordoloi, D., Tripathy, R., Mohapatra, S. K., Sarangi, P. K., and Sharma, P. (2024, September). Convolutional Neural Network Based on TensorFlow for the Recognition of Handwritten Digits in the Odia. In 2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET) (pp. 1-5). IEEE.

[289] Abbass, A. M., and Fyath, R. S. (2024). Enhanced approach for artificial neural network-based optical fiber channel modeling: Geometric constellation shaping WDM system as a case study. Journal of Applied Research and Technology, 22(6), 768-780.

[290] Prabha, D., Subramanian, R. S., Dinesh, M. G., and Girija, P. (2024). Sustainable Farming Through AI-Enabled Precision Agriculture. In Artificial Intelligence for Precision Agriculture (pp. 159-182). Auerbach Publications.

[291] Abdelmadjid, S. A. A. D., and Abdeldjallil, A. I. D. I. (2024, November). Optimized Deep Learning Models For Edge Computing: A Comparative Study on Raspberry PI4 For Real-Time Plant Disease Detection. In 2024 4th International Conference on Embedded and Distributed Systems (EDiS) (pp. 273-278). IEEE.

[292] Mlambo, F. (2024). What are Bayesian Neural Networks?.

[293] Team, G. Y. Bifang: A New Free-Flying Cubic Robot for Space Station.

[294] Tabel, L. (2024). Delay Learning in Spiking.

[295] Naderi, S., Chen, B., Yang, T., Xiang, J., Heaney, C. E., Latham, J. P., ... and Pain, C. C. (2024). A discrete element solution method embedded within a Neural Network. Powder Technology, 448, 120258.

[296] Polaka, S. K. R. (2024). Verifica delle reti neurali per l'apprendimento rinforzato sicuro.

[297] Erdogan, L. E., Kanakagiri, V. A. R., Keutzer, K., and Dong, Z. (2024). Stochastic Communication Avoidance for Recommendation Systems. arXiv preprint arXiv:2411.01611.

[298] Liao, F., Tang, Y., Du, Q., Wang, J., Li, M., and Zheng, J. (2024). Domain Progressive Low-dose CT Imaging using Iterative Partial Diffusion Model. IEEE Transactions on Medical Imaging.

[299] Sekhavat, Y. (2024). Looking for creative basis of artificial intelligence art in the midst of order and chaos based on Nietzsche's theories. Theoretical Principles of Visual Arts.

[300] Cai, H., Yang, Y., Tang, Y., Sun, Z., and Zhang, W. (2025). Shapley value-based class activation mapping for improved explainability in neural networks. The Visual Computer, 1-19.

[301] Na, W. (2024). Rach-Space: Novel Ensemble Learning Method With Applications in Weakly Supervised Learning (Master's thesis, Tufts University).

[302] Khajah, M. M. (2024). Supercharging BKT with Multidimensional Generalizable IRT and Skill Discovery. Journal of Educational Data Mining, 16(1), 233-278.

[303] Zhang, Y., Duan, Z., Huang, Y., and Zhu, F. (2024). Theoretical Bound-Guided Hierarchical VAE for Neural Image Codecs. arXiv preprint arXiv:2403.18535.

[304] Wang, L., and Huang, W. (2025). On the convergence analysis of over-parameterized variational autoencoders: a neural tangent kernel perspective. Machine Learning, 114(1), 15.

[305] Li, C. N., Liang, H. P., Zhao, B. Q., Wei, S. H., and Zhang, X. (2024). Machine learning assisted crystal structure prediction made simple. Journal of Materials Informatics, 4(3), N-A.

[306] Huang, Y. (2024). Research Advanced in Image Generation Based on Diffusion Probability Model. Highlights in Science, Engineering and Technology, 85, 452-456.

[307] Chenebuah, E. T. (2024). Artificial Intelligence Simulation and Design of Energy Materials with Targeted Properties (Doctoral dissertation, Université d'Ottawa— University of Ottawa).

[308] Furth, N., Imel, A., and Zawodzinski, T. A. (2024, November). Graph Encoders for Redox Potentials and Solubility Predictions. In Electrochemical Society Meeting Abstracts prime2024 (No. 3, pp. 344-344). The Electrochemical Society, Inc..

[309] Gong, J., Deng, Z., Xie, H., Qiu, Z., Zhao, Z., and Tang, B. Z. (2025). Deciphering Design of Aggregation-Induced Emission Materials by Data Interpretation. Advanced Science, 12(3), 2411345.

[310] Kim, H., Lee, C. H., and Hong, C. (2024, July). VATMAN: Video Anomaly Transformer for Monitoring Accidents and Nefariousness. In 2024 IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (pp. 1-7). IEEE.

[311] Albert, S. W., Doostan, A., and Schaub, H. (2024). Dimensionality Reduction for Onboard Modeling of Uncertain Atmospheres. Journal of Spacecraft and Rockets, 1-13.

[312] Sharma, D. K., Hota, H. S., and Rababaah, A. R. (2024). Machine Learning for Real World Applications (Doctoral dissertation, Department of Computer Science and Engineering, Indian Institute of Technology Patna).

[313] Li, T., Shi, Z., Dale, S. G., Vignale, G., and Lin, M. Jrystal: A JAX-based Differentiable Density Functional Theory Framework for Materials.

[314] Bieberich, S., Li, P., Ngai, J., Patel, K., Vogt, R., Ranade, P., ... and Stafford, S. (2024). Conducting Quantum Machine Learning Through The Lens of Solving Neural Differential Equations On A Theoretical Fault Tolerant Quantum Computer: Calibration and Benchmarking.

[315] Dagréou, M., Ablin, P., Vaiter, S., and Moreau, T. (2024). How to compute Hessian-vector products?. In The Third Blogpost Track at ICLR 2024.

[316] Lohoff, J., and Neftci, E. (2024). Optimizing Automatic Differentiation with Deep Reinforcement Learning. arXiv preprint arXiv:2406.05027.

[317] Legrand, N., Weber, L., Waade, P. T., Daugaard, A. H. M., Khodadadi, M., Mikuš, N., and Mathys, C. (2024). pyhgf: A neural network library for predictive coding. arXiv preprint arXiv:2410.09206.

[318] Alzás, P. B., and Radev, R. (2024). Differentiable nuclear deexcitation simulation for low energy neutrino physics. arXiv preprint arXiv:2404.00180.

[319] Edenhofer, G., Frank, P., Roth, J., Leike, R. H., Guerdi, M., Scheel-Platz, L. I., ... and Enßlin, T. A. (2024). Re-envisioning numerical information field theory (NIFTy. re): A library for Gaussian processes and variational inference. arXiv preprint arXiv:2402.16683.

[320] Chan, S., Kulkarni, P., Paul, H. Y., and Parekh, V. S. (2024, September). Expanding the Horizon: Enabling Hybrid Quantum Transfer Learning for Long-Tailed Chest X-Ray Classification. In 2024 IEEE International Conference on Quantum Computing and Engineering (QCE) (Vol. 1, pp. 572-582). IEEE.

[321] Ye, H., Hu, Z., Yin, R., Boyko, T. D., Liu, Y., Li, Y., ... and Li, Y. (2025). Electron transfer at birnessite/organic compound interfaces: mechanism, regulation, and two-stage kinetic discrepancy in structural rearrangement and decomposition. Geochimica et Cosmochimica Acta, 388, 253-267.

[322] Khan, M., Ludl, A. A., Bankier, S., Björkegren, J. L., and Michoel, T. (2024). Prediction of causal genes at GWAS loci with pleiotropic gene regulatory effects using sets of correlated instrumental variables. PLoS genetics, 20(11), e1011473.

[323] Ojala, K., and Zhou, C. (2024). Determination of outdoor object distances from monocular thermal images.

[324] Popordanoska, T., and Blaschko, M. (2024). Advancing Calibration in Deep Learning: Theory, Methods, and Applications.

[325] Alfieri, A., Cortes, J. M. P., Pastore, E., Castiglione, C., and Rey, G. M. Z. A Deep Q-Network Approach to Job Shop Scheduling with Transport Resources.

[326] Zanardelli, R. (2025). Statistical learning methods for decision-making, with applications in Industry 4.0.

[327] Norouzi, M., Hosseini, S. H., Khoshnevisan, M., and Moshiri, B. (2025). Applications of pretrained CNN models and data fusion techniques in Unity3D for connected vehicles. Applied Intelligence, 55(6), 390.

[328] Wang, R., Yang, T., Liang, C., Wang, M., and Ci, Y. (2025). Reliable Autonomous Driving Environment Perception: Uncertainty Quantification of Semantic Segmentation. Journal of Transportation Engineering, Part A: Systems, 151(3), 04024117.

[329] Xia, Q., Chen, P., Xu, G., Sun, H., Li, L., and Yu, G. (2024). Adaptive Path-Tracking Controller Embedded With Reinforcement Learning and Preview Model for Autonomous Driving. IEEE Transactions on Vehicular Technology.

[330] Liu, Q., Tang, Y., Li, X., Yang, F., Wang, K., and Li, Z. (2024). MV-STGHAT: Multi-View Spatial-Temporal Graph Hybrid Attention Network for Decision-Making of Connected and Autonomous Vehicles. IEEE Transactions on Vehicular Technology.

[331] Chakraborty, D., and Deka, B. (2025). Deep Learning-based Selective Feature Fusion for Litchi Fruit Detection using Multimodal UAV Sensor Measurements. IEEE Transactions on Artificial Intelligence.

[332] Mirindi, D., Khang, A., and Mirindi, F. (2025). Artificial Intelligence (AI) and Automation for Driving Green Transportation Systems: A Comprehensive Review. Driving Green Transportation System Through Artificial Intelligence and Automation: Approaches, Technologies and Applications, 1-19.

[333] Choudhury, B., Rajakumar, K., Badhale, A. A., Roy, A., Sahoo, R., and Margret, I. N. (2024, June). Comparative Analysis of Advanced Models for Satellite-Based Aircraft Identification. In 2024 International Conference on Smart Systems for Electrical, Electronics, Communication and Computer Engineering (ICSSEECC) (pp. 483-488). IEEE.

[334] Almubarok, W., Rosiani, U. D., and Asmara, R. A. (2024, November). MobileNetV2 Pruning for Improved Efficiency in Catfish Classification on Resource-Limited Devices. In 2024 IEEE 10th Information Technology International Seminar (ITIS) (pp. 271-277). IEEE.

[335] Ding, Q. (2024, February). Classification Techniques of Tongue Manifestation Based on Deep Learning. In 2024 IEEE 3rd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA) (pp. 802-810). IEEE.

[336] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[337] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[338] Sultana, F., Sufian, A., and Dutta, P. (2018, November). Advancements in image classification using convolutional neural network. In 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) (pp. 122-129). IEEE.

[339] Sattler, T., Zhou, Q., Pollefeys, M., and Leal-Taixe, L. (2019). Understanding the limitations of cnn-based absolute camera pose regression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 3302-3312).

[340] Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems.

[341] Nannepagu, M., Babu, D. B., and Madhuri, C. B. Leveraging Hybrid AI Models: DQN, Prophet, BERT, ART-NN, and Transformer-Based Approaches for Advanced Stock Market Forecasting.

[342] De Rose, L., Andresini, G., Appice, A., and Malerba, D. (2024). VINCENT: Cyber-threat detection through vision transformers and knowledge distillation. Computers and Security, 103926.

[343] Buehler, M. J. (2025). Graph-Aware Isomorphic Attention for Adaptive Dynamics in Transformers. arXiv preprint arXiv:2501.02393.

[344] Tabibpour, S. A., and Madanizadeh, S. A. (2024). Solving High-Dimensional Dynamic Programming Using Set Transformer. Available at SSRN 5040295.

[345] Li, S., and Dong, P. (2024, October). Mixed Attention Transformer Enhanced Channel Estimation for Extremely Large-Scale MIMO Systems. In 2024 16th International Conference on Wireless Communications and Signal Processing (WCSP) (pp. 394-399). IEEE.

[346] Asefa, S. H., and Assabie, Y. (2024). Transformer-Based Amharic-to-English Machine Translation with Character Embedding and Combined Regularization Techniques. IEEE Access.

[347] Liao, M., and Chen, M. (2024, November). A new deepfake detection method by vision transformers. In International Conference on Algorithms, High Performance Computing, and Artificial Intelligence (AHPCAI 2024) (Vol. 13403, pp. 953-957). SPIE.

[348] Jiang, L., Cui, J., Xu, Y., Deng, X., Wu, X., Zhou, J., and Wang, Y. (2024, August). SC-Former: Spatial and Channel-wise Transformer with Contrastive Learning for High-Quality PET Image Reconstruction. In 2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM) (pp. 26-31). IEEE.

[349] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... and Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

[350] CHAPPIDI, J., and SUNDARAM, D. M. (2024). DUAL Q-LEARNING WITH GRAPH NEURAL NETWORKS: A NOVEL APPROACH TO ANIMAL DETECTION IN CHALLENGING ECOSYSTEMS. Journal of Theoretical and Applied Information Technology, 102(23).

[351] Joni, R. (2024). Delving into Deep Learning: Illuminating Techniques and Visual Clarity for Image Analysis (No. 12808). EasyChair.

[352] Kalaiarasi, G., Sudharani, B., Jonnalagadda, S. C., Battula, H. V., and Sanagala, B. (2024, July). A Comprehensive Survey of Image Steganography. In 2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS) (pp. 1225-1230). IEEE.

[353] Arjmandi-Tash, A. M., Mansourian, A., Rahsepar, F. R., and Abdi, Y. (2024). Predicting Photodetector Responsivity through Machine Learning. Advanced Theory and Simulations, 2301219.

[354] Gao, Y. (2024). Neural networks meet applied mathematics: GANs, PINNs, and transformers. HKU Theses Online (HKUTO).

[355] Hisama, K., Ishikawa, A., Aspera, S. M., and Koyama, M. (2024). Theoretical Catalyst Screening of Multielement Alloy Catalysts for Ammonia Synthesis Using Machine Learning Potential and Generative Artificial Intelligence. The Journal of Physical Chemistry C, 128(44), 18750-18758.

[356] Wang, M., and Zhang, Y. (2024). Image Segmentation in Complex Backgrounds using an Improved Generative Adversarial Network. International Journal of Advanced Computer Science and Applications, 15(5).

[357] Alonso, N. I., and Arias, F. (2025). The Mathematics of Q-Learning and the Hamilton-Jacobi-Bellman Equation. Fernando, The Mathematics of Q-Learning and the Hamilton-Jacobi-Bellman Equation (January 05, 2025).

[358] Lu, C., Shi, L., Chen, Z., Wu, C., and Wierman, A. (2024). Overcoming the Curse of Dimensionality in Reinforcement Learning Through Approximate Factorization. arXiv preprint arXiv:2411.07591.

[359] Humayoo, M. (2024). Time-Scale Separation in Q-Learning: Extending TD ($\triangle$) for Action-Value Function Decomposition. arXiv preprint arXiv:2411.14019.

[360] Jia, L., Qi, N., Su, Z., Chu, F., Fang, S., Wong, K. K., and Chae, C. B. (2024). Game theory and reinforcement learning for anti-jamming defense in wireless communications: Current research, challenges, and solutions. IEEE Communications Surveys and Tutorials.

[361] Chai, J., Chen, E., and Fan, J. (2025). Deep Transfer Q-Learning for Offline Non-Stationary Reinforcement Learning. arXiv preprint arXiv:2501.04870.

[362] Yao, J., and Gong, X. (2024, October). Communication-Efficient and Resilient Distributed Deep Reinforcement Learning for Multi-Agent Systems. In 2024 IEEE International Conference on Unmanned Systems (ICUS) (pp. 1521-1526). IEEE.

[363] Liu, Y., Yang, T., Tian, L., and Pei, J. (2025). SGD-TripleQNet: An Integrated Deep Reinforcement Learning Model for Vehicle Lane-Change Decision. Mathematics, 13(2), 235.

[364] Masood, F., Ahmad, J., Al Mazroa, A., Alasbali, N., Alazeb, A., and Alshehri, M. S. (2025). Multi IRS-Aided Low-Carbon Power Management for Green Communication in 6G Smart Agriculture Using Deep Game Theory. Computational Intelligence, 41(1), e70022.

[365] Patrick, B. Reinforcement Learning for Dynamic Economic Models.

[366] El Mimouni, I., and Avrachenkov, K. (2025, January). Deep Q-Learning with Whittle Index for Contextual Restless Bandits: Application to Email Recommender Systems. In Northern Lights Deep Learning Conference 2025.

[367] Shefin, R. S., Rahman, M. A., Le, T., and Alqahtani, S. (2024). xSRL: Safety-Aware Explainable Reinforcement Learning–Safety as a Product of Explainability. arXiv preprint arXiv:2412.19311.

[368] Khlifi, A., Othmani, M., and Kherallah, M. (2025). A Novel Approach to Autonomous Driving Using DDQN-Based Deep Reinforcement Learning.

[369] Kuczkowski, D. (2024). Energy efficient multi-objective reinforcement learning algorithm for traffic simulation.

[370] Krauss, R., Zielasko, J., and Drechsler, R. Large-Scale Evolutionary Optimization of Artificial Neural Networks Using Adaptive Mutations.

[371] Ahamed, M. S., Pey, J. J. J., Samarakoon, S. B. P., Muthugala, M. V. J., and Elara, M. R. (2025). Reinforcement Learning for Reconfigurable Robotic Soccer. IEEE Access.

[372] Elmquist, A., Serban, R., and Negrut, D. (2024). A methodology to quantify simulation-vs-reality differences in images for autonomous robots. IEEE Sensors Journal.

[373] Kobanda, A., Portelas, R., Maillard, O. A., and Denoyer, L. (2024). Hierarchical Subspaces of Policies for Continual Offline Reinforcement Learning. arXiv preprint arXiv:2412.14865.

[374] Xu, J., Xie, G., Zhang, Z., Hou, X., Zhang, S., Ren, Y., and Niyato, D. (2025). UPEGSim: An RL-Enabled Simulator for Unmanned Underwater Vehicles Dedicated in the Underwater Pursuit-Evasion Game. IEEE Internet of Things Journal, 12(3), 2334-2346.

[375] Patadiya, K., Jain, R., Moteriya, J., Palaniappan, D., Kumar, P., and Premavathi, T. (2024, December). Application of Deep Learning to Generate Auto Player Mode in Car Based Game. In 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 233-237). IEEE.

[376] Janjua, J. I., Kousar, S., Khan, A., Ihsan, A., Abbas, T., and Saeed, A. Q. (2024, December). Enhancing Scalability in Reinforcement Learning for Open Spaces. In 2024 International Conference on Decision Aid Sciences and Applications (DASA) (pp. 1-8). IEEE.

[377] Yang, L., Li, Y., Wang, J., and Sherratt, R. S. (2020). Sentiment analysis for E-commerce product reviews in Chinese based on sentiment lexicon and deep learning. IEEE access, 8, 23522-23530.

[378] Manikandan, C., Kumar, P. S., Nikitha, N., Sanjana, P. G., and Dileep, Y. Filtering Emails Using Natural Language Processing.

[379] ISIAKA, S. O., BABATUNDE, R. S., and ISIAKA, R. M. Exploring Artificial Intelligence (AI) Technologies in Predictive Medicine: A Systematic Review.

[380] Petrov, A., Zhao, D., Smith, J., Volkov, S., Wang, J., and Ivanov, D. Deep Learning Approaches for Emotional State Classification in Textual Data.

[381] Liang, M. (2025). Leveraging natural language processing for automated assessment and feedback production in virtual education settings. Journal of Computational Methods in Sciences and Engineering, 14727978251314556.

[382] Jin, L. (2025). Research on Optimization Strategies of Artificial Intelligence Algorithms for the Integration and Dissemination of Pharmaceutical Science Popularization Knowledge. Scientific Journal of Technology, 7(1), 45-55.

[383] McNicholas, B. A., Madden, M. G., and Laffey, J. G. (2025). Natural language processing in critical care: opportunities, challenges, and future directions. Intensive Care Medicine, 1-5.

[384] Abd Al Abbas, M., and Khammas, B. M. (2024). Efficient IoT Malware Detection Technique Using Recurrent Neural Network. Iraqi Journal of Information and Communication Technology, 7(3), 29-42.

[385] Kalonia, S., and Upadhyay, A. (2025). Deep learning-based approach to predict software faults. In Artificial Intelligence and Machine Learning Applications for Sustainable Development (pp. 326-348). CRC Press.

[386] Han, S. C., Weld, H., Li, Y., Lee, J., and Poon, J. Natural Language Understanding in Conversational AI with Deep Learning.

[387] Potter, K., and Egon, A. RECURRENT NEURAL NETWORKS (RNNS) FOR TIME SERIES FORECASTING.

[388] Yatkin, M. A., Kõrgesaar, M., and Işlak, Ü. (2025). A Topological Approach to Enhancing Consistency in Machine Learning via Recurrent Neural Networks. Applied Sciences, 15(2), 933.

[389] Saifullah, S. (2024). Comparative Analysis of LSTM and GRU Models for Chicken Egg Fertility Classification using Deep Learning.

[390] Noguer I Alonso, Miquel, The Mathematics of Recurrent Neural Networks (October 27, 2024). Available at SSRN: https://ssrn.com/abstract=5001243 or http://dx.doi.org/10.2139/ssrn.5001243

[391] Tu, Z., Jeffries, S. D., Morse, J., and Hemmerling, T. M. (2024). Comparison of time-series models for predicting physiological metrics under sedation. Journal of Clinical Monitoring and Computing, 1-11.

[392] Zuo, Y., Jiang, J., and Yada, K. (2025). Application of hybrid gate recurrent unit for in-store trajectory prediction based on indoor location system. Scientific Reports, 15(1), 1055.

[393] Lima, R., Scardua, L. A., and De Almeida, G. M. (2024). Predicting Temperatures Inside a Steel Slab Reheating Furnace Using Neural Networks. Authorea Preprints.

[394] Khan, S., Muhammad, Y., Jadoon, I., Awan, S. E., and Raja, M. A. Z. (2025). Leveraging LSTM-SMI and ARIMA architecture for robust wind power plant forecasting. Applied Soft Computing, 112765.

[395] Guo, Z., and Feng, L. (2024). Multi-step prediction of greenhouse temperature and humidity based on temporal position attention LSTM. Stochastic Environmental Research and Risk Assessment, 1-28.

[396] Abdelhamid, N. M., Khechekhouche, A., Mostefa, K., Brahim, L., and Talal, G. (2024). Deep-RNN based model for short-time forecasting photovoltaic power generation using IoT. Studies in Engineering and Exact Sciences, 5(2), e11461-e11461.

[397] Rohman, F. N., and Farikhin, B. S. Hyperparameter Tuning of Random Forest Algorithm for Diabetes Classification.

[398] Rahman, M. Utilizing Machine Learning Techniques for Early Brain Tumor Detection.

[399] Nandi, A., Singh, H., Majumdar, A., Shaw, A., and Maiti, A. Optimizing Baby Sound Recognition using Deep Learning through Class Balancing and Model Tuning.

[400] Sianga, B. E., Mbago, M. C., and Msengwa, A. S. (2025). PREDICTING THE PREVALENCE OF CARDIOVASCULAR DISEASES USING MACHINE LEARNING ALGORITHMS. Intelligence-Based Medicine, 100199.

[401] Li, L., Hu, Y., Yang, Z., Luo, Z., Wang, J., Wang, W., ... and Zhang, Z. (2025). Exploring the assessment of post-cardiac valve surgery pulmonary complication risks through the integration of wearable continuous physiological and clinical data. BMC Medical Informatics and Decision Making, 25(1), 1-11.

[402] Lázaro, F. L., Madeira, T., Melicio, R., Valério, D., and Santos, L. F. (2025). Identifying Human Factors in Aviation Accidents with Natural Language Processing and Machine Learning Models. Aerospace, 12(2), 106.

[403] Li, Z., Zhong, J., Wang, H., Xu, J., Li, Y., You, J., ... and Dev, S. (2025). RAINER: A Robust Ensemble Learning Grid Search-Tuned Framework for Rainfall Patterns Prediction. arXiv preprint arXiv:2501.16900.

[404] Khurshid, M. R., Manzoor, S., Sadiq, T., Hussain, L., Khan, M. S., and Dutta, A. K. (2025). Unveiling diabetes onset: Optimized XGBoost with Bayesian optimization for enhanced prediction. PloS one, 20(1), e0310218.

[405] Kanwar, M., Pokharel, B., and Lim, S. (2025). A new random forest method for landslide susceptibility mapping using hyperparameter optimization and grid search techniques. International Journal of Environmental Science and Technology, 1-16.

[406] Fadil, M., Akrom, M., and Herowati, W. (2025). Utilization of Machine Learning for Predicting Corrosion Inhibition by Quinoxaline Compounds. Journal of Applied Informatics and Computing, 9(1), 173-177.

[407] Emmanuel, J., Isewon, I., and Oyelade, J. (2025). An Optimized Deep-Forest Algorithm Using a Modified Differential Evolution Optimization Algorithm: A Case of Host-Pathogen Protein-Protein Interaction Prediction. Computational and Structural Biotechnology Journal.

[408] Gaurav, A., Gupta, B. B., Attar, R. W., Alhomoud, A., Arya, V., and Chui, K. T. (2025). Driver identification in advanced transportation systems using osprey and salp swarm optimized random forest model. Scientific Reports, 15(1), 2453.

[409] Ning, C., Ouyang, H., Xiao, J., Wu, D., Sun, Z., Liu, B., ... and Huang, G. (2025). Development and validation of an explainable machine learning model for mortality prediction among patients with infected pancreatic necrosis. eClinicalMedicine, 80.

[410] Muñoz, V., Ballester, C., Copaci, D., Moreno, L., and Blanco, D. (2025). Accelerating hyperparameter optimization with a secretary. Neurocomputing, 129455.

[411] Balcan, M. F., Nguyen, A. T., and Sharma, D. (2025). Sample complexity of data-driven tuning of model hyperparameters in neural networks with structured parameter-dependent dual function. arXiv preprint arXiv:2501.13734.

[412] Azimi, H., Kalhor, E. G., Nabavi, S. R., Behbahani, M., and Vardini, M. T. (2025). Data-based modeling for prediction of supercapacitor capacity: Integrated machine learning and metaheuristic algorithms. Journal of the Taiwan Institute of Chemical Engineers, 170, 105996.

[413] Shibina, V., and Thasleema, T. M. (2025). Voice feature-based diagnosis of Parkinson's disease using nature inspired squirrel search algorithm with ensemble learning classifiers. Iran Journal of Computer Science, 1-25.

[414] Chang, F., Dong, S., Yin, H., Ye, X., Wu, Z., Zhang, W., and Zhu, H. (2025). 3D displacement time series prediction of a north-facing reservoir landslide powered by InSAR and machine learning. Journal of Rock Mechanics and Geotechnical Engineering.

[415] Cihan, P. (2025). Bayesian Hyperparameter Optimization of Machine Learning Models for Predicting Biomass Gasification Gases. Applied Sciences, 15(3), 1018.

[416] Makomere, R., Rutto, H., Alugongo, A., Koech, L., Suter, E., and Kohitlhetse, I. (2025). Enhanced dry SO_2 capture estimation using Python-driven computational frameworks with hyperparameter tuning and data augmentation. Unconventional Resources, 100145.

[417] Bakır, H. (2025). A new method for tuning the CNN pre-trained models as a feature extractor for malware detection. Pattern Analysis and Applications, 28(1), 26.

[418] Liu, Y., Yin, H., and Li, Q. (2025). Sound absorption performance prediction of multi-dimensional Helmholtz resonators based on deep learning and hyperparameter optimization. Physica Scripta.

[419] Ma, Z., Zhao, M., Dai, X., and Chen, Y. (2025). Anomaly detection for high-speed machining using hybrid regularized support vector data description. Robotics and Computer-Integrated Manufacturing, 94, 102962.

[420] El-Bouzaidi, Y. E. I., Hibbi, F. Z., and Abdoun, O. (2025). Optimizing Convolutional Neural Network Impact of Hyperparameter Tuning and Transfer Learning. In Innovations in Optimization and Machine Learning (pp. 301-326). IGI Global Scientific Publishing.

[421] Mustapha, B., Zhou, Y., Shan, C., and Xiao, Z. (2025). Enhanced Pneumonia Detection in Chest X-Rays Using Hybrid Convolutional and Vision Transformer Networks. Current Medical Imaging, e15734056326685.

[422] Adly, S., and Attouch, H. (2024). Complexity Analysis Based on Tuning the Viscosity Parameter of the Su-Boyd-Candès Inertial Gradient Dynamics. Set-Valued and Variational Analysis, 32(2), 17.

[423] Wang, Z., and Peypouquet, J. G. Nesterov's Accelerated Gradient Method for Strongly Convex Functions: From Inertial Dynamics to Iterative Algorithms.

[424] Hermant, J., Renaud, M., Aujol, J. F., and Rondepierre, C. D. A. (2024). Nesterov momentum for convex functions with interpolation: is it faster than Stochastic gradient descent?. Book of abstracts PGMO DAYS 2024, 68.

[425] Alavala, S., and Gorthi, S. (2024). 3D CBCT Challenge 2024: Improved Cone Beam CT Reconstruction using SwinIR-Based Sinogram and Image Enhancement. arXiv preprint arXiv:2406.08048.

[426] Li, C. J. (2024). Unified Momentum Dynamics in Stochastic Gradient Optimization. Available at SSRN 4981009.

[427] Gupta, K., and Wojtowytsch, S. (2024). Nesterov acceleration in benignly non-convex landscapes. arXiv preprint arXiv:2410.08395.

[428] Razzouki, O. F., Charroud, A., El Allali, Z., Chetouani, A., and Aslimani, N. (2024, December). A Survey of Advanced Gradient Methods in Machine Learning. In 2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet) (pp. 1-7). IEEE.

[429] Wang, J., Du, B., Su, Z., Hu, K., Yu, J., Cao, C., ... and Guo, H. (2025). A fast LMS-based digital background calibration technique for 16-bit SAR ADC with modified shuffling scheme. Microelectronics Journal, 156, 106547.

[430] Naeem, K., Bukhari, A., Daud, A., Alsahfi, T., Alshemaimri, B., and Alhajlah, M. (2024). Machine Learning and Deep Learning Optimization Algorithms for Unconstrained Convex Optimization Problem. IEEE Access.

[431] Campos, C. M., de Diego, D. M., and Torrente, J. (2024). Momentum-based gradient descent methods for Lie groups. arXiv preprint arXiv:2404.09363.

[432] Jing Li, Hewan Chen, Mohd Shahizan Othman, Naomie Salim, Lizawati Mi Yusuf, Shamini Raja Kumaran, NFIoT-GATE-DTL IDS: Genetic algorithm-tuned ensemble of deep transfer learning for NetFlow-based intrusion detection system for internet of things, Engineering Applications of Artificial Intelligence, Volume 143, 2025, 110046, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2025.110046.

[433] GÜL, M.F., Bakır, H. GA-ML: enhancing the prediction of water electrical conductivity through genetic algorithm-based end-to-end hyperparameter tuning. Earth Sci Inform 18, 191 (2025). https://doi.org/10.1007/s12145-024-01610-1

[434] Sen, A., Sen, U., Paul, M., Padhy, A. P., Sai, S., Mallik, A., and Mallick, C. (2025). QGAPHEnsemble: Combining Hybrid QLSTM Network Ensemble via Adaptive Weighting for Short Term Weather Forecasting. arXiv preprint arXiv:2501.10866.

[435] Roy, A., Sen, A., Gupta, S., Haldar, S., Deb, S., Vankala, T. N., and Das, A. (2025). DeepEyeNet: Adaptive Genetic Bayesian Algorithm Based Hybrid ConvNeXtTiny Framework For Multi-Feature Glaucoma Eye Diagnosis. arXiv preprint arXiv:2501.11168.

[436] Jiang, T., Lu, W., Lu, L., Xu, L., Xi, W., Liu, J., and Zhu, Y. (2025). Inlet Passage Hydraulic Performance Optimization of Coastal Drainage Pump System Based on Machine Learning Algorithms. Journal of Marine Science and Engineering, 13(2), 274.

[437] Borah, J., and Chandrasekaran, M. (2025). Application of Machine Learning-Based Approach to Predict and Optimize Mechanical Properties of Additively Manufactured Polyether Ether Ketone Biopolymer Using Fused Deposition Modeling. Journal of Materials Engineering and Performance, 1-17.

[438] Tan, Q., He, D., Sun, Z., Yao, Z., zhou, J. X., and Chen, T. (2025). A deep reinforcement learning based metro train operation control optimization considering energy conservation and passenger comfort. Engineering Research Express.

[439] García-Galindo, A., López-De-Castro, M., and Armañanzas, R. (2025). Fair prediction sets through multi-objective hyperparameter optimization. Machine Learning, 114(1), 27.

[440] Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. Advances in neural information processing systems, 27.

[441] Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with ReLU activation function.

[442] Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. Neural networks, 94, 103-114.

[443] Telgarsky, M. (2016, June). Benefits of depth in neural networks. In Conference on learning theory (pp. 1517-1539). PMLR.

[444] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. Advances in neural information processing systems, 30.

[445] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3), 107-115.

[446] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.

[447] Kipf, T. N., and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[448] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in neural information processing systems, 30.

[449] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[450] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks?. arXiv preprint arXiv:1810.00826.

[451] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017, July). Neural message passing for quantum chemistry. In International conference on machine learning (pp. 1263-1272). PMLR.

[452] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261.

[453] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

[454] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 974-983).

[455] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... and Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.

[456] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378, 686-707.

[457] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. Nature Reviews Physics, 3(6), 422-440.

[458] Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. SIAM review, 63(1), 208-228.

[459] Sirignano, J., and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. Journal of computational physics, 375, 1339-1364.

[460] Wang, S., Teng, Y., and Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5), A3055-A3081.

[461] Mishra, S., and Molinaro, R. (2023). Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA Journal of Numerical Analysis, 43(1), 1-43.

[462] Zhang, D., Guo, L., and Karniadakis, G. E. (2020). Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. SIAM Journal on Scientific Computing, 42(2), A639-A665.

[463] Jin, X., Cai, S., Li, H., and Karniadakis, G. E. (2021). NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. Journal of Computational Physics, 426, 109951.

[464] Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. (2020). Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express, 28(8), 11618-11633.

[465] Psichogios, D. C., and Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling. AIChE Journal, 38(10), 1499-1511.

[466] Chizat, L., and Bach, F. (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. Advances in neural information processing systems, 31.

[467] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. (2019, May). Gradient descent finds global minima of deep neural networks. In International conference on machine learning (pp. 1675-1685). PMLR.

[468] Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. (2019, May). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In International Conference on Machine Learning (pp. 322-332). PMLR.

[469] Allen-Zhu, Z., Li, Y., and Song, Z. (2019, May). A convergence theory for deep learning via over-parameterization. In International conference on machine learning (pp. 242-252). PMLR.

[470] Cao, Y., and Gu, Q. (2019). Generalization bounds of stochastic gradient descent for wide and deep neural networks. Advances in neural information processing systems, 32.

[471] Yang, G. (2019). Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. arXiv preprint arXiv:1902.04760.

[472] Huang, J., and Yau, H. T. (2020, November). Dynamics of deep neural networks and neural tangent hierarchy. In International conference on machine learning (pp. 4542-4551). PMLR.

[473] Belkin, M., Ma, S., and Mandal, S. (2018, July). To understand deep learning we need to understand kernel learning. In International Conference on Machine Learning (pp. 541-549). PMLR.

[474] Sra, S., Nowozin, S., and Wright, S. J. (Eds.). (2011). Optimization for machine learning. Mit Press.

[475] Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015, February). The loss surfaces of multilayer networks. In Artificial intelligence and statistics (pp. 192-204). PMLR.

[476] Arora, S., Cohen, N., and Hazan, E. (2018, July). On the optimization of deep networks: Implicit acceleration by overparameterization. In International conference on machine learning (pp. 244-253). PMLR.

[477] Baratin, A., George, T., Laurent, C., Hjelm, R. D., Lajoie, G., Vincent, P., and Lacoste-Julien, S. (2020). Implicit regularization in deep learning: A view from function space. arXiv preprint arXiv:2008.00938.

[478] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. (2018, July). The mechanics of n-player differentiable games. In International Conference on Machine Learning (pp. 354-363). PMLR.

[479] Han, J., and Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in mathematics and statistics, 5(4), 349-380.

[480] Beck, C., Becker, S., Grohs, P., Jaafari, N., and Jentzen, A. (2021). Solving the Kolmogorov PDE by means of deep learning. Journal of Scientific Computing, 88, 1-28.

[481] Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34), 8505-8510.

[482] Jentzen, A., Salimova, D., and Welti, T. (2018). A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. arXiv preprint arXiv:1809.07321.

[483] Yu, B. (2018). The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1), 1-12.

[484] Khoo, Y., Lu, J., and Ying, L. (2021). Solving parametric PDE problems with artificial neural networks. European Journal of Applied Mathematics, 32(3), 421-435.

[485] Hutzenthaler, M., and Kruse, T. (2020). Multilevel Picard approximations of high-dimensional semilinear parabolic differential equations with gradient-dependent nonlinearities. SIAM Journal on Numerical Analysis, 58(2), 929-961.

[486] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(185), 1-52.

[487] Falkner, S., Klein, A., and Hutter, F. (2018, July). BOHB: Robust and efficient hyperparameter optimization at scale. In International conference on machine learning (pp. 1437-1446). PMLR.

[488] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., ... and Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2, 230-246.

[489] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.

[490] Slivkins, A., Zhou, X., Sankararaman, K. A., and Foster, D. J. (2024). Contextual Bandits with Packing and Covering Constraints: A Modular Lagrangian Approach via Regression. Journal of Machine Learning Research, 25(394), 1-37.

[491] Hazan, E., Klivans, A., and Yuan, Y. (2017). Hyperparameter optimization: A spectral approach. arXiv preprint arXiv:1706.00764.

[492] Domhan, T., Springenberg, J. T., and Hutter, F. (2015, June). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Twenty-fourth international joint conference on artificial intelligence.

[493] Agrawal, T. (2021). Hyperparameter optimization in machine learning: make your machine learning and deep learning models more efficient (pp. 109-129). New York, NY, USA:: Apress.

[494] Shekhar, S., Bansode, A., and Salim, A. (2021, December). A comparative study of hyperparameter optimization tools. In 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE) (pp. 1-6). IEEE.

[495] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24.

[496] Zoph, B. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

[497] Maclaurin, D., Duvenaud, D., and Adams, R. (2015, June). Gradient-based hyperparameter optimization through reversible learning. In International conference on machine learning (pp. 2113-2122). PMLR.

[498] Pedregosa, F. (2016, June). Hyperparameter optimization with approximate gradient. In International conference on machine learning (pp. 737-746). PMLR.

[499] Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018, July). Bilevel programming for hyperparameter optimization and meta-learning. In International conference on machine learning (pp. 1568-1577). PMLR.

[500] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017, July). Forward and reverse gradient-based hyperparameter optimization. In International Conference on Machine Learning (pp. 1165-1173). PMLR.

[501] Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.

[502] Lorraine, J., Vicol, P., and Duvenaud, D. (2020, June). Optimizing millions of hyperparameters by implicit differentiation. In International conference on artificial intelligence and statistics (pp. 1540-1552). PMLR.

[503] Liang, J., Gonzalez, S., Shahrzad, H., and Miikkulainen, R. (2021, June). Regularized evolutionary population-based training. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 323-331).

[504] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., ... and Kavukcuoglu, K. (2017). Population based training of neural networks. arXiv preprint arXiv:1711.09846.

[505] Co-Reyes, J. D., Miao, Y., Peng, D., Real, E., Levine, S., Le, Q. V., ... and Faust, A. (2021). Evolving reinforcement learning algorithms. arXiv preprint arXiv:2101.03958.

[506] Song, C., Ma, Y., Xu, Y., and Chen, H. (2024). Multi-population evolutionary neural architecture search with stacked generalization. Neurocomputing, 587, 127664.

[507] Wan, X., Lu, C., Parker-Holder, J., Ball, P. J., Nguyen, V., Ru, B., and Osborne, M. (2022, September). Bayesian generational population-based training. In International conference on automated machine learning (pp. 14-1). PMLR.

[508] García-Valdez, M., Mancilla, A., Castillo, O., and Merelo-Guervós, J. J. (2023). Distributed and asynchronous population-based optimization applied to the optimal design of fuzzy controllers. Symmetry, 15(2), 467.

[509] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 2623-2631).

[510] Akiba, T., Shing, M., Tang, Y., Sun, Q., and Ha, D. (2025). Evolutionary optimization of model merging recipes. Nature Machine Intelligence, 1-10.

[511] Kadhim, Z. S., Abdullah, H. S., and Ghathwan, K. I. (2022). Artificial Neural Network Hyperparameters Optimization: A Survey. International Journal of Online and Biomedical Engineering, 18(15).

[512] Jeba, J. A. (2021). Case study of Hyperparameter optimization framework Optuna on a Multi-column Convolutional Neural Network (Doctoral dissertation, University of Saskatchewan).

[513] Yang, L., and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295-316.

[514] Wang, T. (2024). Multi-objective hyperparameter optimisation for edge machine learning.

[515] Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

[516] Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). Automated machine learning: methods, systems, challenges (p. 219). Springer Nature.

[517] Jamieson, K., and Talwalkar, A. (2016, May). Non-stochastic best arm identification and hyperparameter optimization. In Artificial intelligence and statistics (pp. 240-248). PMLR.

[518] Schmucker, R., Donini, M., Zafar, M. B., Salinas, D., and Archambeau, C. (2021). Multi-objective asynchronous successive halving. arXiv preprint arXiv:2106.12639.

[519] Dong, X., Shen, J., Wang, W., Shao, L., Ling, H., and Porikli, F. (2019). Dynamical hyperparameter optimization via deep reinforcement learning in tracking. IEEE transactions on pattern analysis and machine intelligence, 43(5), 1515-1529.

[520] Rijsdijk, J., Wu, L., Perin, G., and Picek, S. (2021). Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(3), 677-707.

[521] Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. (2019). Reinforcement learning for neural architecture search: A review. Image and Vision Computing, 89, 57-66.

[522] Afshar, R. R., Zhang, Y., Vanschoren, J., and Kaymak, U. (2022). Automated reinforcement learning: An overview. arXiv preprint arXiv:2201.05000.

[523] Wu, J., Chen, S., and Liu, X. (2020). Efficient hyperparameter optimization through model-based reinforcement learning. Neurocomputing, 409, 381-393.

[524] Iranfar, A., Zapater, M., and Atienza, D. (2021). Multiagent reinforcement learning for hyperparameter optimization of convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(4), 1034-1047.

[525] He, X., Zhao, K., and Chu, X. (2021). AutoML: A survey of the state-of-the-art. Knowledge-based systems, 212, 106622.

[526] Gomaa, I., Zidane, A., Mokhtar, H. M., and El-Tazi, N. (2022). SML-AutoML: A Smart Meta-Learning Automated Machine Learning Framework.

[527] Khan, A. N., Khan, Q. W., Rizwan, A., Ahmad, R., and Kim, D. H. (2025). Consensus-Driven Hyperparameter Optimization for Accelerated Model Convergence in Decentralized Federated Learning. Internet of Things, 30, 101476.

[528] Morrison, N., and Ma, E. Y. (2025). Efficiency of machine learning optimizers and meta-optimization for nanophotonic inverse design tasks. APL Machine Learning, 3(1).

[529] Berdyshev, D. A., Grachev, A. M., Shishkin, S. L., and Kozyrskiy, B. L. (2024). EEG-Reptile: An Automatized Reptile-Based Meta-Learning Library for BCIs. arXiv preprint arXiv:2412.19725.

[530] Pratellesi, C. (2025). Meta Learning for Flow Cytometry Cell Classification (Doctoral dissertation, Technische Universität Wien).

[531] García, C. A., Gil-de-la-Fuente, A., Barbas, C., and Otero, A. (2022). Probabilistic metabolite annotation using retention time prediction and meta-learned projections. Journal of Cheminformatics, 14(1), 33.

[532] Deng, L., Raissi, M., and Xiao, M. (2024). Meta-Learning-Based Surrogate Models for Efficient Hyperparameter Optimization. Authorea Preprints.

[533] Jae, J., Hong, J., Choo, J., and Kwon, Y. D. (2024). Reinforcement learning to learn quantum states for Heisenberg scaling accuracy. arXiv preprint arXiv:2412.02334.

[534] Upadhyay, R., Phlypo, R., Saini, R., and Liwicki, M. (2025). Meta-Sparsity: Learning Optimal Sparse Structures in Multi-task Networks through Meta-learning. arXiv preprint arXiv:2501.12115.

[535] Paul, S., Ghosh, S., Das, D., and Sarkar, S. K. (2025). Advanced Methodologies for Optimal Neural Network Design and Performance Enhancement. In Nature-Inspired Optimization Algorithms for Cyber-Physical Systems (pp. 403-422). IGI Global Scientific Publishing.

[536] Egele, R., Mohr, F., Viering, T., and Balaprakash, P. (2024). The unreasonable effectiveness of early discarding after one epoch in neural network hyperparameter optimization. Neurocomputing, 127964.

[537] Wojciuk, M., Swiderska-Chadaj, Z., Siwek, K., and Gertych, A. (2024). Improving classification accuracy of fine-tuned CNN models: Impact of hyperparameter optimization. Heliyon, 10(5).

[538] Geissler, D., Zhou, B., Suh, S., and Lukowicz, P. (2024). Spend More to Save More (SM2): An Energy-Aware Implementation of Successive Halving for Sustainable Hyperparameter Optimization. arXiv preprint arXiv:2412.08526.

[539] Hosseini Sarcheshmeh, A., Etemadfard, H., Najmoddin, A., and Ghalehnovi, M. (2024). Hyperparameters' role in machine learning algorithm for modeling of compressive strength of recycled aggregate concrete. Innovative Infrastructure Solutions, 9(6), 212.

[540] Sankar, S. U., Dhinakaran, D., Selvaraj, R., Verma, S. K., Natarajasivam, R., and Kishore, P. P. (2024). Optimizing diabetic retinopathy disease prediction using PNAS, ASHA, and transfer learning. In Advances in Networks, Intelligence and Computing (pp. 62-71). CRC Press.

[541] Zhang, X., and Duh, K. (2024, September). Best Practices of Successive Halving on Neural Machine Translation and Large Language Models. In Proceedings of the 16th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track) (pp. 130-139).

[542] Aach, M., Sarma, R., Neukirchen, H., Riedel, M., and Lintermann, A. (2024). Resource-Adaptive Successive Doubling for Hyperparameter Optimization with Large Datasets on High-Performance Computing Systems. arXiv preprint arXiv:2412.02729.

[543] Jang, D., Yoon, H., Jung, K., and Chung, Y. D. (2024). QHB+: Accelerated Configuration Optimization for Automated Performance Tuning of Spark SQL Applications. IEEE Access.

[544] Chen, Y., Wen, Z., Chen, J., and Huang, J. (2024, May). Enhancing the Performance of Bandit-based Hyperparameter Optimization. In 2024 IEEE 40th International Conference on Data Engineering (ICDE) (pp. 967-980). IEEE.

[545] Zhang, Y., Wu, H., and Yang, Y. (2024). FlexHB: a More Efficient and Flexible Framework for Hyperparameter Optimization. arXiv preprint arXiv:2402.13641.

[546] Srivastava, N. (2013). Improving neural networks with dropout. University of Toronto, 182(566), 7.

[547] Baldi, P., and Sadowski, P. J. (2013). Understanding dropout. Advances in neural information processing systems, 26.

[548] Gal, Y., and Ghahramani, Z. (2016, June). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059). PMLR.

[549] Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. Advances in neural information processing systems, 30.

[550] Gal, Y., and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. Advances in neural information processing systems, 29.

[551] Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33, 1-22.

[552] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society Series B: Statistical Methodology, 58(1), 267-288.

[553] Meinshausen, N. (2007). Relaxed lasso. Computational Statistics and Data Analysis, 52(1), 374-393.

[554] Carvalho, C. M., Polson, N. G., and Scott, J. G. (2009, April). Handling sparsity via the horseshoe. In Artificial intelligence and statistics (pp. 73-80). PMLR.

[555] Hoerl, A. E., and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

[556] Cesa-Bianchi, N., Conconi, A., and Gentile, C. (2004). On the generalization ability of on-line learning algorithms. IEEE Transactions on Information Theory, 50(9), 2050-2057.

[557] Devroye, L., Györfi, L., and Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science and Business Media.

[558] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H. T. (2012). Learning from data (Vol. 4, p. 4). New York: AMLBook.

[559] Shalev-Shwartz, S., and Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

[560] Bühlmann, P., and Van De Geer, S. (2011). Statistics for high-dimensional data: methods, theory and applications. Springer Science and Business Media.

[561] Gareth, J., Daniela, W., Trevor, H., and Robert, T. (2013). An introduction to statistical learning: with applications in R. Spinger.

[562] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression.

[563] Fan, J., and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96(456), 1348-1360.

[564] Meinshausen, N., and Bühlmann, P. (2006). High-dimensional graphs and variable selection with the lasso.

[565] Montavon, G., Orr, G., and Müller, K. R. (Eds.). (2012). Neural networks: tricks of the trade (Vol. 7700). springer.

[566] Prechelt, L. (2002). Early stopping-but when?. In Neural Networks: Tricks of the trade (pp. 55-69). Berlin, Heidelberg: Springer Berlin Heidelberg.

[567] Brownlee, J. (2019). Develop deep learning models on theano and TensorFlow using keras. J Chem Inf Model, 53(9), 1689-1699.

[568] Zhang, H. (2017). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.

[569] Shorten, C., and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1), 1-48.

[570] Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

[571] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501.

[572] Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78-87.

[573] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. Journal of the royal statistical society: Series B (Methodological), 36(2), 111-133.

[574] LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. Advances in neural information processing systems, 2.

[575] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

[576] Frankle, J., and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.

[577] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28.

[578] Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018). Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270.

[579] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.

[580] Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2020). Pruning neural networks at initialization: Why are we missing the mark?. arXiv preprint arXiv:2009.08576.

[581] Breiman, L. (1996). Bagging predictors. Machine learning, 24, 123-140.

[582] Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.

[583] Freund, Y., and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), 119-139.

[584] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of statistics, 1189-1232.

[585] Zhou, Z. H. (2025). Ensemble methods: foundations and algorithms. CRC press.

[586] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine learning, 40, 139-157.

[587] Chen, T., and Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[588] Bühlmann, P., and Yu, B. (2003). Boosting with the L 2 loss: regression and classification. Journal of the American Statistical Association, 98(462), 324-339.

[589] Hinton, G. E., and Van Camp, D. (1993, August). Keeping the neural networks simple by minimizing the description length of the weights. In Proceedings of the sixth annual conference on Computational learning theory (pp. 5-13).

[590] Bishop, C. M. (1995). Training with noise is equivalent to Tikhonov regularization. Neural computation, 7(1), 108-116.

[591] Grandvalet, Y., and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. Advances in neural information processing systems, 17.

[592] Wager, S., Wang, S., and Liang, P. S. (2013). Dropout training as adaptive regularization. Advances in neural information processing systems, 26.

[593] Pei, Z., Zhang, Z., Chen, J., Liu, W., Chen, B., Huang, Y., ... and Lu, Y. (2025). KAN–CNN: A Novel Framework for Electric Vehicle Load Forecasting with Enhanced Engineering Applicability and Simplified Neural Network Tuning. Electronics, 14(3), 414.

[594] Chen, H. (2024). Augmenting image data using noise, rotation and shifting.

[595] An, D., Liu, P., Feng, Y., Ding, P., Zhou, W., and Yu, B. (2024). Dynamic weighted knowledge distillation for brain tumor segmentation. Pattern Recognition, 155, 110731.

[596] SONG, Y. F., and LIU, Y. (2024). Fast adversarial training method based on data augmentation and label noise. Journal of Computer Applications, 0.

[597] Hosseini, S. A., Servaes, S., Rahmouni, N., Therriault, J., Tissot, C., Macedo, A. C., ... and Rosa-Neto, P. (2024). Leveraging T1 MRI Images for Amyloid Status Prediction in Diverse Cognitive Conditions Using Advanced Deep Learning Models. Alzheimer's and Dementia, 20, e094153.

[598] Cakmakci, U. B. Deep Learning Approaches for Pediatric Bone Age Prediction from Hand Radiographs.

[599] Surana, A. V., Pawar, S. E., Raha, S., Mali, N., and Mukherjee, T. (2024). ENSEMBLE FINE TUNED MULTI LAYER PERCEPTRON FOR PREDICTIVE ANALYSIS OF WEATHER PATTERNS AND RAINFALL FORECASTING FROM SATELLITE DATA. ICTACT Journal on Soft Computing, 15(2).

[600] Chanda, A. An In-Depth Analysis of CIFAR-100 Using Inception v3.

[601] Zaitoon, R., Mohanty, S. N., Godavarthi, D., and Ramesh, J. V. N. (2024). SPBTGNS: Design of an Efficient Model for Survival Prediction in Brain Tumour Patients using Generative Adversarial Network with Neural Architectural Search Operations. IEEE Access.

[602] Bansal, A., Sharma, D. R., and Kathuria, D. M. Bayesian-Optimized Ensemble Approach for Fall Detection: Integrating Pose Estimation with Temporal Convolutional and Graph Neural Networks. Available at SSRN 4974349.

[603] Kusumaningtyas, E. M., Ramadijanti, N., and Rijal, I. H. K. (2024, August). Convolutional Neural Network Implementation with MobileNetV2 Architecture for Indonesian Herbal Plants Classification in Mobile App. In 2024 International Electronics Symposium (IES) (pp. 521-527). IEEE.

[604] Yadav, A. C., Alam, Z., and Mufeed, M. (2024, August). U-Net-Driven Advancements in Breast Cancer Detection and Segmentation. In 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT) (Vol. 1, pp. 1-6). IEEE.

[605] Alshamrani, A. F. A., and Alshomran, F. (2024). Optimizing Breast Cancer Mammogram Classification through a Dual Approach: A Deep Learning Framework Combining ResNet50, SMOTE, and Fully Connected Layers for Balanced and Imbalanced Data. IEEE Access.

[606] Zamindar, N. (2024). Using Artificial Intelligence for Thermographic Image Analysis: Applications to the Arc Welding Process (Doctoral dissertation, Politecnico di Torino).

[607] Xu, M., Yin, H., and Zhong, S. (2024, July). Enhancing Generalization and Convergence in Neural Networks through a Dual-Phase Regularization Approach with Excitatory-Inhibitory Transition. In 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET (pp. 1-4). IEEE.

[608] Elshamy, R., Abu-Elnasr, O., Elhoseny, M., and Elmougy, S. (2024). Enhancing colorectal cancer histology diagnosis using modified deep neural networks optimizer. Scientific Reports, 14(1), 19534.

[609] Vinay, K., Kodipalli, A., Swetha, P., and Kumaraswamy, S. (2024, May). Analysis of prediction of pneumonia from chest X-ray images using CNN and transfer learning. In 2024 5th International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.

[610] Gai, S., and Huang, X. (2024). Regularization method for reduced biquaternion neural network. Applied Soft Computing, 166, 112206.

[611] Xu, Y. (2025). Deep regularization techniques for improving robustness in noisy record linkage task. Advances in Engineering Innovation, 15, 9-13.

[612] Liao, Z., Li, S., Zhou, P., and Zhang, C. (2025). Decay regularized stochastic configuration networks with multi-level data processing for UAV battery RUL prediction. Information Sciences, 701, 121840.

[613] Dong, Z., Yang, C., Li, Y., Huang, L., An, Z., and Xu, Y. (2024, May). Class-wise Image Mixture Guided Self-Knowledge Distillation for Image Classification. In 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD) (pp. 310-315). IEEE.

[614] Ba, Y., Mancenido, M. V., and Pan, R. (2024). How Does Data Diversity Shape the Weight Landscape of Neural Networks?. arXiv preprint arXiv:2410.14602.

[615] Li, Z., Zhang, Y., and Li, W. (2024, September). Fusion of L2 Regularisation and Hybrid Sampling Methods for Multi-Scale SincNet Audio Recognition. In 2024 IEEE 7th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 7, pp. 1556-1560). IEEE.

[616] Zang, X., and Yan, A. (2024, May). A Stochastic Configuration Network with Attenuation Regularization and Multi-kernel Learning and Its Application. In 2024 36th Chinese Control and Decision Conference (CCDC) (pp. 2385-2390). IEEE.

[617] Moradi, R., Berangi, R., and Minaei, B. (2020). A survey of regularization strategies for deep models. Artificial Intelligence Review, 53(6), 3947-3986.

[618] Rodríguez, P., Gonzalez, J., Cucurull, G., Gonfaus, J. M., and Roca, X. (2016). Regularizing cnns with locally constrained decorrelations. arXiv preprint arXiv:1611.01967.

[619] Tian, Y., and Zhang, Y. (2022). A comprehensive survey on regularization strategies in machine learning. Information Fusion, 80, 146-166.

[620] Cong, Y., Liu, J., Fan, B., Zeng, P., Yu, H., and Luo, J. (2017). Online similarity learning for big data with overfitting. IEEE Transactions on Big Data, 4(1), 78-89.

[621] Salman, S., and Liu, X. (2019). Overfitting mechanism and avoidance in deep neural networks. arXiv preprint arXiv:1901.06566.

[622] Wang, K., Muthukumar, V., and Thrampoulidis, C. (2021). Benign overfitting in multiclass classification: All roads lead to interpolation. Advances in Neural Information Processing Systems, 34, 24164-24179.

[623] Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., ... and Mhaskar, H. (2017). Theory of deep learning III: explaining the non-overfitting puzzle. arXiv preprint arXiv:1801.00173.

[624] Oyedotun, O. K., Olaniyi, E. O., and Khashman, A. (2017). A simple and practical review of over-fitting in neural network learning. International Journal of Applied Pattern Recognition, 4(4), 307-328.

[625] Luo, X., Chang, X., and Ban, X. (2016). Regression and classification using extreme learning machine based on L1-norm and L2-norm. Neurocomputing, 174, 179-186.

[626] Zhou, Y., Yang, Y., Wang, D., Zhai, Y., Li, H., and Xu, Y. (2024). Innovative Ghost Channel Spatial Attention Network with Adaptive Activation for Efficient Rice Disease Identification. Agronomy, 14(12), 2869.

[627] Omole, O. J., Rosa, R. L., Saadi, M., and Rodriguez, D. Z. (2024). AgriNAS: Neural Architecture Search with Adaptive Convolution and Spatial–Time Augmentation Method for Soybean Diseases. AI, 5(4), 2945-2966.

[628] Tripathi, L., Dubey, P., Kalidoss, D., Prasad, S., Sharma, G., and Dubey, P. (2024, December). Deep Learning Approaches for Brain Tumour Detection Using VGG-16 Architecture. In 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 256-261). IEEE.

[629] Singla, S., and Gupta, R. (2024, December). Pneumonia Detection from Chest X-Ray Images Using Transfer Learning with EfficientNetB1. In 2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS) (pp. 894-899). IEEE.

[630] Al-Adhaileh, M. H., Alsharbi, B. M., Aldhyani, T., Ahmad, S., Almaiah, M., Ahmed, Z. A., ... and Singh, S. DLAAD-Deep Learning Algorithms Assisted Diagnosis of Chest Disease Using Radiographic Medical Images. Frontiers in Medicine, 11, 1511389.

[631] Harvey, E., Petrov, M., and Hughes, M. C. (2025). Learning Hyperparameters via a Data-Emphasized Variational Objective. arXiv preprint arXiv:2502.01861.

[632] Mahmood, T., Saba, T., Al-Otaibi, S., Ayesha, N., and Almasoud, A. S. (2025). AI-Driven Microscopy: Cutting-Edge Approach for Breast Tissue Prognosis Using Microscopic Images. Microscopy Research and Technique.

[633] Shen, Q. (2025). Predicting the value of football players: machine learning techniques and sensitivity analysis based on FIFA and real-world statistical datasets. Applied Intelligence, 55(4), 265.

[634] Guo, X., Wang, M., Xiang, Y., Yang, Y., Ye, C., Wang, H., and Ma, T. (2025). Uncertainty Driven Adaptive Self-Knowledge Distillation for Medical Image Segmentation. IEEE Transactions on Emerging Topics in Computational Intelligence.

[635] Zambom, A. Z., and Dias, R. (2013). A review of kernel density estimation with applications to econometrics. International Econometric Review, 5(1), 20-42.

[636] Reyes, M., Francisco-Fernández, M., and Cao, R. (2016). Nonparametric kernel density estimation for general grouped data. Journal of Nonparametric Statistics, 28(2), 235-249.

[637] Tenreiro, C. (2024). A Parzen–Rosenblatt type density estimator for circular data: exact and asymptotic optimal bandwidths. Communications in Statistics-Theory and Methods, 53(20), 7436-7452.

[638] Devroye, L., and Penrod, C. S. (1984). The consistency of automatic kernel density estimates. The Annals of Statistics, 1231-1249.

[639] El Machkouri, M. (2011). Asymptotic normality of the Parzen–Rosenblatt density estimator for strongly mixing random fields. Statistical Inference for Stochastic Processes, 14, 73-84.

[640] Slaoui, Y. (2018). Bias reduction in kernel density estimation. Journal of Nonparametric Statistics, 30(2), 505-522.

[641] Michalski, A. (2016). The use of kernel estimators to determine the distribution of groundwater level. Meteorology Hydrology and Water Management. Research and Operational Applications, 4(1), 41-46.

[642] Gramacki, A., and Gramacki, A. (2018). Kernel density estimation. Nonparametric Kernel Density Estimation and Its Computational Aspects, 25-62.

[643] Desobry, F., Davy, M., and Fitzgerald, W. J. (2007, April). Density kernels on unordered sets for kernel-based signal processing. In 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07 (Vol. 2, pp. II-417). IEEE.

[644] Gasser, T., and Müller, H. G. (1979). Kernel estimation of regression functions. In Smoothing Techniques for Curve Estimation: Proceedings of a Workshop held in Heidelberg, April 2–4, 1979 (pp. 23-68). Springer Berlin Heidelberg.

[645] Gasser, T., and Müller, H. G. (1984). Estimating regression functions and their derivatives by the kernel method. Scandinavian journal of statistics, 171-185.

[646] Härdle, W., and Gasser, T. (1985). On robust kernel estimation of derivatives of regression functions. Scandinavian journal of statistics, 233-240.

[647] Müller, H. G. (1987). Weighted local regression and kernel methods for nonparametric curve fitting. Journal of the American Statistical Association, 82(397), 231-238.

[648] Chu, C. K. (1993). A new version of the Gasser-Mueller estimator. Journal of Nonparametric Statistics, 3(2), 187-193.

[649] Peristera, P., and Kostaki, A. (2005). An evaluation of the performance of kernel estimators for graduating mortality data. Journal of Population Research, 22, 185-197.

[650] Müller, H. G. (1991). Smooth optimum kernel estimators near endpoints. Biometrika, 78(3), 521-530.

[651] Gasser, T., Gervini, D., Molinari, L., Hauspie, R. C., and Cameron, N. (2004). Kernel estimation, shape-invariant modelling and structural analysis. Cambridge Studies in Biological and Evolutionary Anthropology, 179-204.

[652] Jennen-Steinmetz, C., and Gasser, T. (1988). A unifying approach to nonparametric regression estimation. Journal of the American Statistical Association, 83(404), 1084-1089.

[653] Müller, H. G. (1997). Density adjusted kernel smoothers for random design nonparametric regression. Statistics and probability letters, 36(2), 161-172.

[654] Neumann, M. H., and Thorarinsdottir, T. L. (2006). Asymptotic minimax estimation in nonparametric autoregression. Mathematical Methods of Statistics, 15(4), 374.

[655] Steland, A. THE AVERAGE RUN LENGTH OF KERNEL CONTROL CHARTS FOR DEPENDENT TIME SERIES.

[656] Makkulau, A. T. A., Baharuddin, M., and Agusrawati, A. T. P. M. (2023, December). Multivariable Semiparametric Regression Used Priestley-Chao Estimators. In Proceedings of the 5th International Conference on Statistics, Mathematics, Teaching, and Research 2023 (ICSMTR 2023) (Vol. 109, p. 118). Springer Nature.

[657] Staniswalis, J. G. (1989). The kernel estimate of a regression function in likelihood-based models. Journal of the American Statistical Association, 84(405), 276-283.

[658] Mack, Y. P., and Müller, H. G. (1988). Convolution type estimators for nonparametric regression. Statistics and probability letters, 7(3), 229-239.

[659] Jones, M. C., Davies, S. J., and Park, B. U. (1994). Versions of kernel-type regression estimators. Journal of the American Statistical Association, 89(427), 825-832.

[660] Ghosh, S. (2015). Surface estimation under local stationarity. Journal of Nonparametric Statistics, 27(2), 229-240.

[661] Liu, C. W., and Luor, D. C. (2023). Applications of fractal interpolants in kernel regression estimations. Chaos, Solitons and Fractals, 175, 113913.

[662] Agua, B. M., and Bouzebda, S. (2024). Single index regression for locally stationary functional time series. AIMS Math, 9, 36202-36258.

[663] Bouzebda, S., Nezzal, A., and Elhattab, I. (2024). Limit theorems for nonparametric conditional U-statistics smoothed by asymmetric kernels. AIMS Mathematics, 9(9), 26195-26282.

[664] Zhao, H., Qian, Y., and Qu, Y. (2025). Mechanical performance degradation modelling and prognosis method of high-voltage circuit breakers considering censored data. IET Science, Measurement and Technology, 19(1), e12235.

[665] Patil, M. D., Kannaiyan, S., and Sarate, G. G. (2024). Signal denoising based on bias-variance of intersection of confidence interval. Signal, Image and Video Processing, 18(11), 8089-8103.

[666] Kakani, K., and Radhika, T. S. L. (2024). Nonparametric and nonlinear approaches for medical data analysis. International Journal of Data Science and Analytics, 1-19.

[667] Kato, M. (2024). Debiased Regression for Root-N-Consistent Conditional Mean Estimation. arXiv preprint arXiv:2411.11748.

[668] Sadek, A. M., and Mohammed, L. A. (2024). Evaluation of the Performance of Kernel Nonparametric Regression and Ordinary Least Squares Regression. JOIV: International Journal on Informatics Visualization, 8(3), 1352-1360.

[669] Gong, A., Choi, K., and Dwivedi, R. (2024). Supervised Kernel Thinning. arXiv preprint arXiv:2410.13749.

[670] Zavatone-Veth, J. A., and Pehlevan, C. (2025). Nadaraya–Watson kernel smoothing as a random energy model. Journal of Statistical Mechanics: Theory and Experiment, 2025(1), 013404.

[671] Ferrigno, S. (2024, December). Nonparametric estimation of reference curves. In CMStatistics 2024.

[672] Fan, X., Leng, C., and Wu, W. (2025). Causal Inference under Interference: Regression Adjustment and Optimality. arXiv preprint arXiv:2502.06008.

[673] Atanasov, A., Bordelon, B., Zavatone-Veth, J. A., Paquette, C., and Pehlevan, C. (2025). Two-Point Deterministic Equivalence for Stochastic Gradient Dynamics in Linear Models. arXiv preprint arXiv:2502.05074.

[674] Mishra, U., Gupta, D., Sarkar, A., and Hazarika, B. B. (2025). A hybrid approach for plant leaf detection using ResNet50-intuitionistic fuzzy RVFL (ResNet50-IFRVFLC) classifier. Computers and Electrical Engineering, 123, 110135.

[675] Elsayed, M. M., and Nazier, H. (2025). Technology and evolution of occupational employment in Egypt (1998–2018): a task-based framework. Review of Economics and Political Science.

[676] Kong, X., Li, C., and Pan, Y. (2025). Association Between Heavy Metals Mixtures and Life's Essential 8 Score in General US Adults. Cardiovascular Toxicology, 1-12.

[677] Bracale, D., Banerjee, M., Sun, Y., Stoll, K., and Turki, S. (2025). Dynamic Pricing in the Linear Valuation Model using Shape Constraints. arXiv preprint arXiv:2502.05776.

[678] Köhne, F., Philipp, F. M., Schaller, M., Schiela, A., and Worthmann, K. (2024). L∞-error bounds for approximations of the Koopman operator by kernel extended dynamic mode decomposition. arXiv preprint arXiv:2403.18809.

[679] Sadeghi, R., and Beyeler, M. (2025). Efficient Spatial Estimation of Perceptual Thresholds for Retinal Implants via Gaussian Process Regression. arXiv preprint arXiv:2502.06672.

[680] Naresh, E., Patil, A., and Bhuvan, S. (2025, February). Enhancing network security with eBPF-based firewall and machine learning. In Data Science and Exploration in Artificial Intelligence: Proceedings of the First International Conference On Data Science and Exploration in Artificial Intelligence (CODE-AI 2024) Bangalore, India, 3rd-4th July, 2024 (Volume 1) (p. 169). CRC Press.

[681] Zhao, W., Chen, H., Liu, T., Tuo, R., and Tian, C. From Deep Additive Kernel Learning to Last-Layer Bayesian Neural Networks via Induced Prior Approximation. In The 28th International Conference on Artificial Intelligence and Statistics.