Unified Deep Learning

Wei Wang*

Abstract

With the increase in depth and complexity, deep learning networks have made significant progress in various directions. However, the theoretical understanding of deep learning is still incomplete. Early research successfully proved the universal approximation theorem for linear networks, but these proofs were limited to linear networks. Subsequent studies attempted to prove the approximation properties of convolutions and Transformers, but the proof processes often relied on complex assumptions or were very intricate. This paper aims to propose a unified approach to demonstrate that multi-layer networks composed of convolutions and Transformers are specific realizations of the universal approximation theorem. This approach does not require any assumptions and proves that most networks composed of convolutions and Transformers can be mathematically written in the same form as the fully proven universal approximation theorem, thus establishing them as specific implementations of the universal approximation theorem. This bridges the gap between deep learning practice and theoretical understanding. The method of unifying them is to represent these network architectures (linear, convolutional, and Transformer) in matrix-vector form, hence this unified approach is called the matrix-vector method. This paper takes an important step towards unifying the entire field of deep learning. It deepens our theoretical understanding and reveals the fundamental principles behind the exceptional performance of these networks. It also paves the way for exploring new research directions and optimizing the learning process in various deep learning applications.

Keywords: Universal Approximation, Explainable Neural Network

1. Introduction

Deep learning, as the most significant research direction in artificial intelligence, has achieved remarkable advancements in machine intelligence, although it currently lacks general human-like intelligence. However, it has demonstrated extraordinary capabilities and promising future applications in specific domains such as image processing, natural language processing, and temporal analysis. Various deep learning models have been designed, which are essentially stacks of multiple layers of Linear, 1D, 2D, 3D convolutions or Transformers.

Given the outstanding performance of deep learning in AI, a multitude of researchers are devoting themselves to model development, aiming to find efficient, high-performance models that require less data. Despite this, deep learning remains largely empirical, with researchers often designing network architectures based on intuition or previous experience, testing their performance, and creating different models for specific problems. This approach consumes substantial human resources and computational power and is inherently speculative. To address this issue, there is a pressing need for a fundamental theoretical framework for deep learning – one that aligns with current experiments and experiences while guiding future developments.

This paper proposes a unifying approach to deep learning and, based on this, provides answers to some fundamental questions, offering valuable theoretical references for future advancements in the

*Corresponding author.
Email address: weiwangnnn@yeah.net (Wei Wang)

field. The challenge in establishing a theory within the realm of deep learning lies in the complexity of its networks and models. Due to the versatility of modules (Linear, 1D, 2D, 3D convolutions, or Transformers) in deep learning, designing models resembles building with toy blocks, leading to an array of diverse designs. While researchers can often provide subjective explanations for certain parts of well-performing models, this diversity complicates efforts to explain or theorize about individual models systematically. Without a unified theory and with subjective interpretations abound, disagreements arise among researchers regarding the same model or module, without any effective means to prove or disprove each other's perspectives.

The aim of this paper is to present a unified theoretical perspective for various deep learning models, capable of explaining issues across these models. A prerequisite for this is to unify the basic modules: Linear, 1D, 2D, 3D convolutions, or Transformers under the same mathematical form, providing a foundation for theoretical inquiry. To achieve this, the paper introduces the matrix-vector method, which embodies a unifying principle for these basic modules by transforming them into a form of parameter matrices multiplied by vectors. This matrix-vector formulation, when combined with activation functions across multiple layers, aligns with the Universal Approximation Theorem. The theorem serves as the initial theoretical cornerstone of deep learning, demonstrating that a single layer of linear network with activation functions can approximate any continuous function over a closed interval, and later findings showed that multi-layered networks also conform to this theorem. Thus, they are collectively referred to as manifestations of the Universal Approximation Theory. Consequently, networks constructed from multiple layers of these basic modules (with activation functions) are indeed concrete implementations of the Universal Approximation Theory.

Additionally, several issues play a pivotal role in shaping the development of deep learning, including parameter redundancy, generalization capability, performance bottlenecks, and the future direction of deep learning. This paper employs the Universal Approximation Theorem to provide mathematically sound explanations for these critical challenges.

In summary, this article's contributions can be highlighted in three main aspects:

- I have introduced a matrix-vector approach, which serves as a unified method for various fundamental deep learning modules, such as Linear, Convolution, and Transformer. Through the utilization of this approach, we are able to represent these modules using a common mathematical framework. This signifies that we can intuitively compare the distinctions among these modules from a mathematical perspective, facilitating an analysis of their strengths and weaknesses. Consequently, this approach lays the foundation for effective model design. Furthermore, by leveraging this approach, I can demonstrate that the majority of deep learning models are specific implementations of the Universal Approximation Theorem.
- When using the matrix-vector method, I discover that the differences between linear, Transformer, and convolutional modules primarily lie in their receptive fields.
- I address some common issues in deep learning, including parameter redundancy, generalization performance, interpretability, performance bottlenecks, and the future development of deep learning, based on the Universal Approximation Theorem. I provide a more reliable mathematical explanation for these problems.

2. The Basic Format of Deep Learning

In the domain of deep learning, an array of fundamental modules and activation functions constitutes its core. Given the expansive diversity of modern deep learning networks, it is impractical to enumerate them all; hence, we exemplify with ResNet, a highly prevalent architecture that has

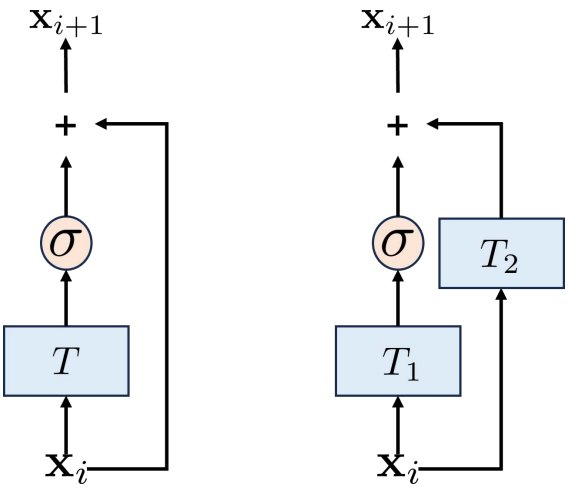

Figure 1: The general format of ResNet block.

served as the basis for numerous other networks, to illustrate the essential structure of a deep learning network. Figure 1 depicts the configuration of a basic block within ResNet. If the number of channels alters, we utilize the left component corresponding to Eq. 1, while in the absence of such change, we employ the right component aligned with Eq. 2.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \sigma(T(\mathbf{x}_i)) \tag{1}$$

$$\mathbf{x}_{i+1} = T_2(\mathbf{x}_i) + \sigma(T_1(\mathbf{x}_i)) \tag{2}$$

The above equations elucidate two typical building blocks found within ResNet. Here, $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ respectively denote the input and output to the i^{th} layer of the network, whereas the transformations T_1 and T_2 embody fundamental operations inherent to deep learning, and σ signifies an activation function, often exemplified by ReLU.

3. The Universal Approximation Theory

We aim to explore various deep learning models within the framework of the Universal Approximation Theorem. To begin with, we provide a concise overview of this theorem, originally presented in [1], which encompasses numerous conclusions and proof details. This paper specifically focuses on the form of the Universal Approximation Theorem as exemplified in [1].

Theorem 2 from [1] states that if σ is any continuous sigmoidal function, then finite sums of the following form:

$$G(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j \sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x} + \theta_j\right) \tag{3}$$

are dense in $C\left(\mathbf{I}_n\right)$. Here, $\mathbf{W}_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed. For any $f \in C\left(\mathbf{I}_n\right)$ and $\varepsilon > 0$, there exists a sum $G(\mathbf{x})$ of the above form for which:

$$|G(\mathbf{x}) - f(\mathbf{x})| < \varepsilon \quad \text{for all} \quad \mathbf{x} \in \mathbf{I}_n. \tag{4}$$

This implies that, when N is sufficiently large, a single-layer neural network can approximate any continuous function on a closed interval. [2] further demonstrates that multilayer feedforward networks also conform to the Universal Approximation Theorem, capable of approximating arbitrary Borel measurable functions. However, in these proofs, the theorem does not directly apply to

convolutional and Transformer networks. Nevertheless, if we can reformulate the mathematical expressions of Linear, Convolutional, and Transformer architectures into the format of $\mathbf{Wx}$, where $\mathbf{W}$ represents the parameter matrix and $\mathbf{x}$ is a column vector, it suggests that most conclusions about the Universal Approximation Theorem are equally applicable to multi-layer networks composed of these components.

Consequently, our objective is to represent Linear, Convolutional, and Transformer architectures as a matrix-vector multiplication similar to E.q.5. This approach offers two key benefits: Firstly, it allows for a fundamental mathematical investigation of deep learning models. Secondly, it provides a more intuitive and objective basis for comparing differences among various models. On this foundation, we propose the Matrix-Vector method to transform Linear, Convolutional, and Transformer structures into the $\mathbf{Wx}$ format.

4. Matrix-Vector Method and Deep Learning

In the field of deep learning, we encounter a landscape dominated by three main network architectures: linear, transformer, and convolutional networks. Although these networks are touted for their capabilities, it is important to note that their advantages in this regard often rely on intuitive assumptions. To this end, in this section, I propose a new approach that goes beyond these intuitive boundaries. By leveraging the transformative power of matrix-vector methods, we seek to unify these different architectures under a comprehensive framework. Our approach promises to go beyond intuition and provide a rigorous framework to examine and understand the true advantages and limitations of each method. With the matrix-vector method as the cornerstone, we begin to explore systematically revealing the strengths and weaknesses of each network prototype. Our aim is to provide a deeper understanding of these networks, rather than relying on superficial assumptions. By adopting this unified approach, we hope that the field of deep learning will move towards a new era of evidence-based decision-making. Our aim is to provide a comprehensive perspective for researchers and practitioners, enabling them to make informed choices when selecting network architectures for specific challenges.

4.1. Format and Advantages of Matrix-Vector Methods

The key to the Matrix-Vector Method lies in matrix multiplication. The introduction of the Matrix-Vector Method in this paper aims to unify and simplify the mathematical representations of linear, Transformer, and convolution operations, transforming them into a unified Matrix-Vector format. The Matrix-Vector Method is a conceptual approach that unifies various transformations using a similar idea, facilitating the coordination of complex computations like Transformers and convolutions.

This method involves two crucial steps: **reorganizing input data** and **rearranging and constructing parameter matrices**.

For clarity, let's first provide a general description of this method. The Matrix-Vector Method can be represented as follows:

$$\mathbf{y} = T(\mathbf{x}) \rightarrow \mathbf{y}' = \mathbf{W}'\mathbf{x}' \tag{5}$$

In this equation, T represents fundamental modules in the field of deep learning, including linear, convolutional, and Transformer networks. $\mathbf{W}'$ is the parameter matrix generated based on T using **rearranging and constructing parameter matrices**. $\mathbf{x}$ denotes the input to T, while $\mathbf{x}'$ corresponds to $\mathbf{x}$ reorganized along the column direction using **reorganizing input data**, and $\mathbf{y}'$ corresponds to $\mathbf{y}$ reorganized along the column direction using **reorganizing input data**.

For convenience, unless specified otherwise, we may slightly abuse symbols, using $\mathbf{x}$ and $\mathbf{y}$ to represent inputs and outputs. Additionally, by default, matrix variables in the original formulas are represented in bold, such as $\mathbf{x}$, and in the Matrix-Vector form, corresponding variables are denoted with a prime symbol (') in the upper right corner, such as $\mathbf{x}'$. Elements within matrices are represented by corresponding lowercase letters with subscripts, for example: x_i.

Reorganizing Input Data: Input data typically appears in 1D, 2D or 3D forms. The goal is to reconstruct them into a unified 1D column vector. For 1D data, it inherently conforms to the format of a 1D column vector and requires no additional modification. However, when dealing with 2D data, we employ a systematic approach. This involves systematically extracting elements from each row, assembling them in order, and forming a 1D column vector. More specifically, this process entails traversing the rows and extracting elements in their order of appearance, then vertically arranging these elements to establish the desired 1D column vector structure. A similar method is used for reorganizing output data. This process can be understood as the transformation from $\mathbf{x}$ to $\mathbf{x}'$ and $\mathbf{y}$ to $\mathbf{y}'$.

Rearranging and Constructing Parameter Matrices: The parameter matrix is a critical element in deep learning, originating from various basic modules like convolutional and Transformer modules. The essence of this stage is to reconfigure the intrinsic parameters of these fundamental modules and subsequently generate the corresponding parameter matrix $\mathbf{W}'$, enabling $\mathbf{W}'$ to satisfy Eq.5. This ingeniously designed relationship allows the generated parameter matrix $\mathbf{W}'$ to be multiplied by the restructured input data $\mathbf{x}'$, producing the reorganized output data $\mathbf{y}'$. The intricate interplay among the parameter matrix, input data transformation, and output data generation forms the basis of this comprehensive work.

The Matrix-Vector Method is an innovative approach aimed at studying fundamental building blocks like linear, Transformer, and convolution within a unified framework. The elegance of the Matrix-Vector Method lies in its ability to encapsulate diverse modules into a single, comprehensible format. This unified representation not only simplifies the understanding of these building blocks but also provides a crucial foundation for comparative analysis. By transforming complex models into a common format, this method prompts a nuanced exploration of their shared attributes and distinctions. Furthermore, since linear, convolutional, and Transformer networks rewritten using the Matrix-Vector Method exhibit nearly identical mathematical forms to Eq.3, almost all multilayer networks with activation functions that can be expressed in the form of Eq. 5 are essentially variations of the Universal Approximation Theory. We provide an example in Section 3. Additionally, due to the computational complexity involved in linear, convolutional, and Transformer modules, we propose a novel computational method called the Diamond Multiplication Method.

4.2. Diamond Matrix Multiplication

The actual computation process of matrices often involves multiplying row vectors by column vectors, which can be complex from the perspectives of computation, observation, and parameter matrix construction. To simplify the construction and observation of parameters for different modules and facilitate the computation process, we introduce the Diamond Matrix Multiplication denoted by the symbol $\diamond$.

Let $\mathbf{W} \in \mathbb{R}^{(m,n)}$ and $\mathbf{x} \in \mathbb{R}^{(m,1)}$. The Diamond Matrix Multiplication is defined as $\mathbf{y} = \mathbf{W} \diamond \mathbf{x}$, where $\mathbf{y} \in \mathbb{R}^{(m,1)}$. The computation procedure involves element-wise multiplication and summation of corresponding elements from left to right across the columns of matrix $\mathbf{W}$ with vector $\mathbf{x}$. The sum of the multiplication of the ith column of $\mathbf{W}$ with $\mathbf{x}$ is assigned as the ith element of $\mathbf{y}$. A detailed calculation process is provided in Section 6. The computation process for $\mathbf{y} = \mathbf{x} \diamond \mathbf{W}$ is identical to that of $\mathbf{W} \diamond \mathbf{x}$, implying that $\mathbf{W} \diamond \mathbf{x} = \mathbf{x} \diamond \mathbf{W}$. However, it should be noted that the Diamond Matrix Multiplication does not possess the properties of associativity and distributivity.

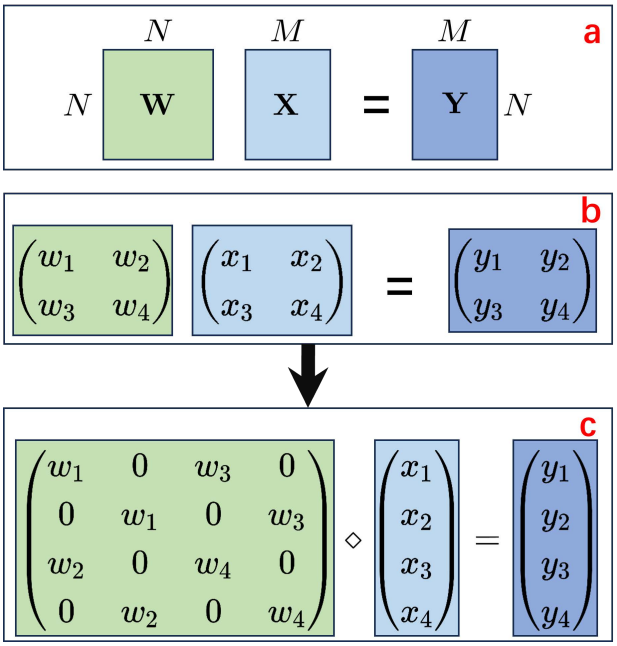

Figure 2: Linear Transformation: This figure demonstrates how to convert a linear transformation into the corresponding matrix-vector form. Part **a**: General linear transformation. Part **b**: A simple example of a linear transformation. Part **c**: Transforming the linear transformation from Part **b** into Matrix-Vector form.

Furthermore, based on its computation method, it can be inferred that the Diamond Matrix Multiplication is related to conventional matrix multiplication, specifically: $\mathbf{W} \diamond \mathbf{x} = \mathbf{W}^T\mathbf{x}$. Additionally, when $\mathbf{W}_1$ and $\mathbf{W}_2$ are square matrices, we can derive the following relationships: $\mathbf{W}_1 \diamond [\mathbf{x} \diamond \mathbf{W}_2] = \mathbf{W}_2^T \diamond \mathbf{W}_1 \diamond \mathbf{x} = \mathbf{W}_{12}\mathbf{x}$ and $[\mathbf{W}_1 \diamond \mathbf{x}] \diamond \mathbf{W}_2 = \mathbf{W}_1^T \diamond \mathbf{W}_2 \diamond \mathbf{x}$, where $\mathbf{W}_{12} = \mathbf{W}_1^T\mathbf{W}_2^T$. Detailed derivations can be found in Section 6.

The Diamond Matrix Multiplication will facilitate a more straightforward representation of linear, convolutional, and Transformer operations in the form of matrix multiplications, while also aiding the analysis of their parameter variations. By adopting this innovative technique, we aim to enhance the intuitive understanding and comparison of these fundamental building blocks in deep learning.

$$\begin{aligned} &\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & & w_{m,n} \end{pmatrix} \diamond \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \\ &= \begin{pmatrix} w_1x_1 + w_{31} * x_2 + \cdots w_{m,1}x_m \\ w_{1,2}x_1 + w_{2,2}x_2 + \cdots w_{m,2}x_m \\ \vdots \\ w_{1,n}x_1 + w_{2,n}x_2 + \cdots w_{mn}x_m \end{pmatrix} \end{aligned} \tag{6}$$

4.3. Matrix-Vector Method for Linear

Our goal is to analyze various transformations by unifying them in the same form, starting from the simplest linear transformation. Subsequent transformations will follow a similar approach. We aim to use the matrix-vector method to transform linear transformations into the form of matrix multiplication by a vector. When the input data is single-channel, no additional operations

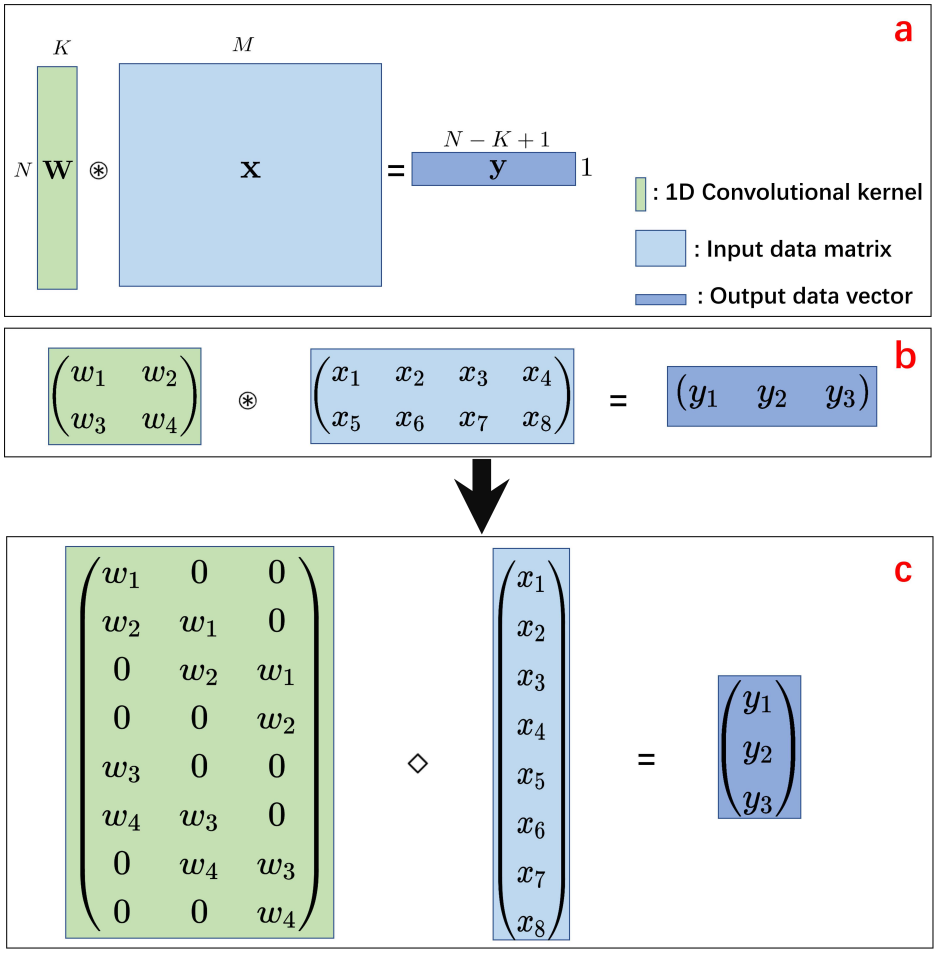

Figure 3: Single-channel Output 1D Convolution: This figure illustrates the scenario where a single convolutional kernel (green box) convolves with the input data (blue box) to produce a single-channel output (deep blue box). (**a**) Conceptual diagram of 1D convolution, using different colored boxes to represent data and parameters. (**b**) Simple example of 1D convolution. (**c**) Matrix-vector representation corresponding to the 1D convolution in part (**b**).

are needed. However, when the input data is multi-channel, we need to perform matrix-vector transformations.

Figure 2 below illustrates this process: Part **a** demonstrates a linear transformation with multi-channel input: $\mathbf{W}\mathbf{x} = \mathbf{y}$. Part **b** provides a concrete example. Part **c** transforms the linear transformation from Part **a** into its corresponding matrix-vector representation: $\mathbf{W}'\mathbf{x}' = \mathbf{y}'$.

Thus, the linear transformation can be represented in matrix-vector form:

$$\begin{aligned} \mathbf{x}^{i+1} = \mathbf{W}_i\mathbf{x}^i \rightarrow \mathbf{x}i + 1' &= \mathbf{W}_i' \diamond \mathbf{x}_i' \\ &= (\mathbf{W}_i')^T\mathbf{x}_i' \end{aligned} \tag{7}$$

Here, $\mathbf{x}^i \in \mathbb{R}^{(N,M)}$ and $\mathbf{x}^{i+1} \in \mathbb{R}^{(N,M)}$ represent the input and output of layer i respectively, while $\mathbf{W}_i \in \mathbb{R}^{(N,N)}$ represents the parameters of layer i. $\mathbf{x}_i'$, $\mathbf{x}_{i+1}'$, and $\mathbf{W}_i'$ are generated based on $\mathbf{x}^i$, $\mathbf{x}^{i+1}$, and $\mathbf{W}_i$ using the matrix-vector method. The derivation of the general form can be found in 6.

Assuming the column direction represents the series dimension and the row direction represents the feature dimension, Figure 3 illustrates the learning approach of linear transformations in a series problem. In this case, all features in the same row share the same parameters, while different rows

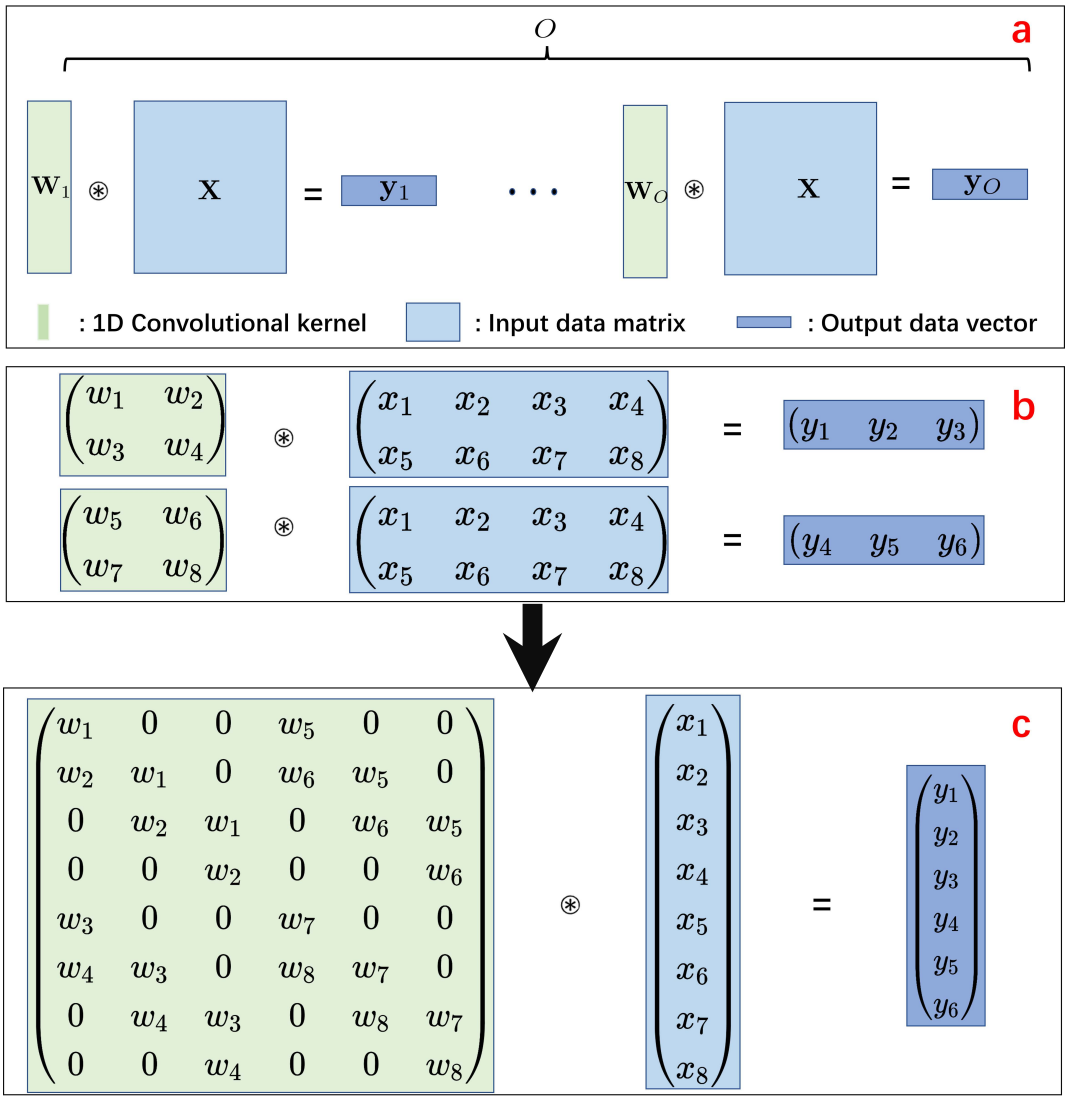

Figure 4: Multi-channel Output 1D Convolution: This figure illustrates the scenario where multiple convolutional kernels (green boxes) convolve with the input data (blue box) to produce O channel outputs (deep blue boxes). (**a**) Conceptual diagram of 1D convolution, using differently colored boxes to represent data and parameters. (**b**) Simple example corresponding to part (**a**). (**c**) Matrix-vector representation corresponding to the 1D convolution in part (**b**).

have different parameters. The issue with this approach lies in whether the patterns of all features are consistent. If they are inconsistent, sharing the same parameters for all features may lead to underfitting for certain features, or it may require learning a compromise solution that adapts to all features. This could potentially hinder the accurate representation and modeling of the dynamics of individual features.

4.4. Matrix-Vector Method for Convolution

Convolution plays a crucial foundational role in the field of deep learning. In this section, we will delve into the intricate relationship between convolution and matrix multiplication.

Concerning 1D convolution, part **a** of Figure 3 vividly illustrates the basic process of single-channel output in 1D convolution: $\mathbf{W} \circledast \mathbf{x} = \mathbf{y}$. Part **b** provides a concise example of single-channel output in one-dimensional convolution. Part **c** further demonstrates how to use the matrix-

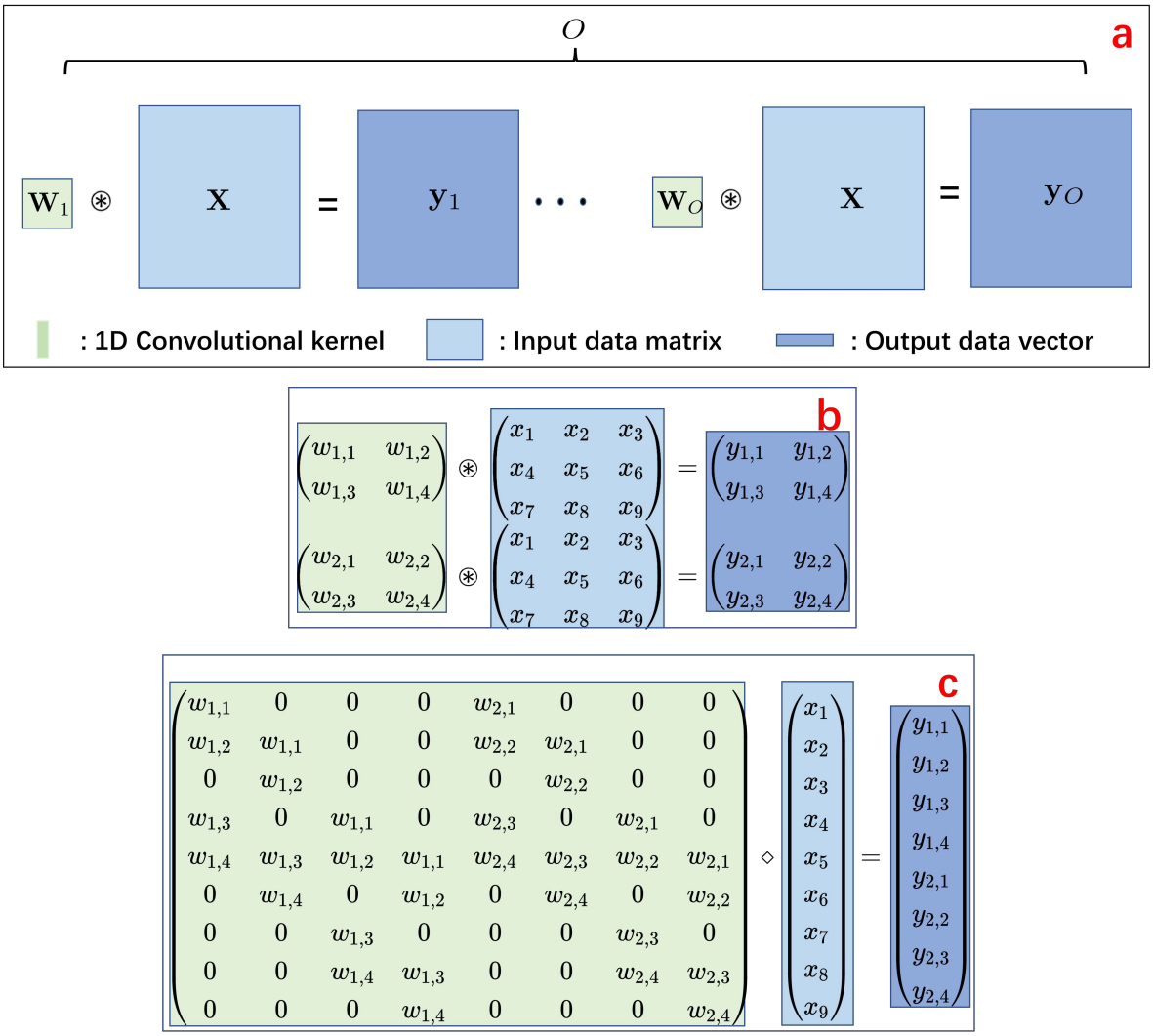

Figure 5: The current illustration demonstrates the procedure for converting a 2D convolution operation featuring a single-channel input and multiple-channel output into a matrix-vector format. In this context, part **a** articulates the standard definition of such a 2D convolutional process. Subsequently, part **b** provides an easily comprehensible instance of this operation. Finally, part **c** reveals the matrix-vector manifestation that parallels the case illustrated in part **b**.

vector method to transform the convolution of single-channel output into the form of diamond multiplication:$\mathbf{W}' \diamond \mathbf{x}' = \mathbf{y}'$, where $\mathbf{x}'$,$\mathbf{W}'$ and $\mathbf{y}'$ are generated from $\mathbf{x}$,$\mathbf{W}$ and $\mathbf{y}$.

Part **a** of Figure 4 elaborates on the fundamental process of 1D convolution with multiple convolution kernels and multiple-channel output:$\mathbf{W}_1 \circledast \mathbf{x} = \mathbf{y}_1 ... \mathbf{W}_O \circledast \mathbf{x} = \mathbf{y}_O$. Part **b** offers a simplified example of part **a**, and part **c** employs the matrix-vector method to convert the convolution of multiple-channel output into the form of diamond multiplication:$\mathbf{W}' \diamond \mathbf{x} = \mathbf{y}'$, where $\mathbf{W}'$ is generated from $\mathbf{W}_1 ... \mathbf{W}_O$ and $\mathbf{y}'$ is generated from $\mathbf{y}_1 ... \mathbf{y}_O$. With the relationship between diamond multiplication and matrix multiplication, we can derive the following formula:

$$\begin{aligned} \mathbf{x}^{i+1} = \mathbf{x}^i \circledast \mathbf{W}_i \rightarrow \mathbf{x}i + 1' &= \mathbf{W}_i' \diamond \mathbf{x}_i' \\ &= (\mathbf{W}_i')^T \mathbf{x}_i' \end{aligned} \tag{8}$$

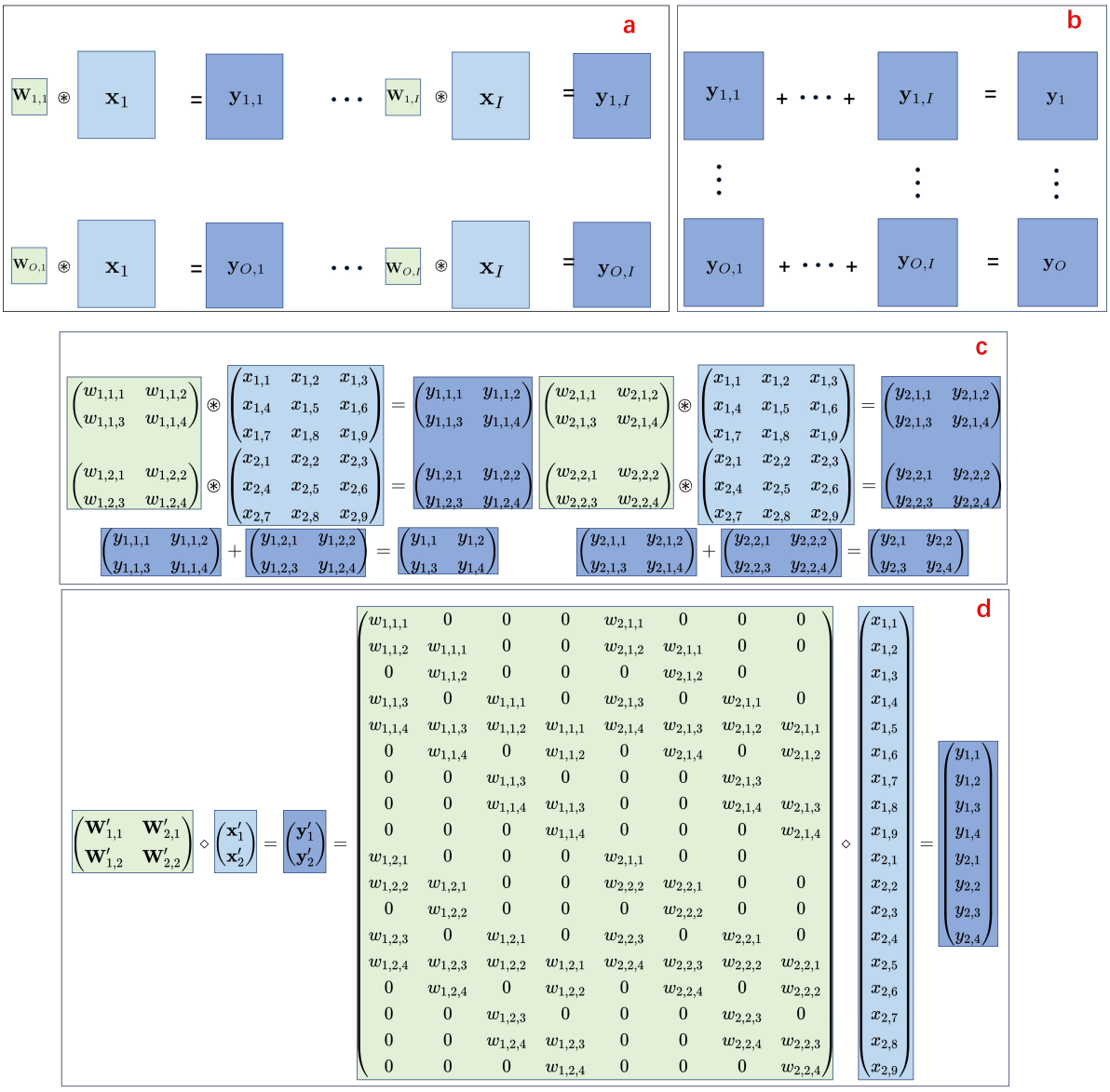

Figure 6: The current diagram elucidates the methodology behind transforming a 2D convolution process that accepts multi-channel input and generates multi-channel output into a matrix-vector format. Divided into parts **a** and **b**, it explicates the overarching principle of such a multichannel 2D convolution operation. Moving forward, part **c** showcases an easily digestible instance of this multi-channel convolutional process. Lastly, part **d** reveals the equivalent matrix-vector representation that corresponds to the example presented in part **c**.

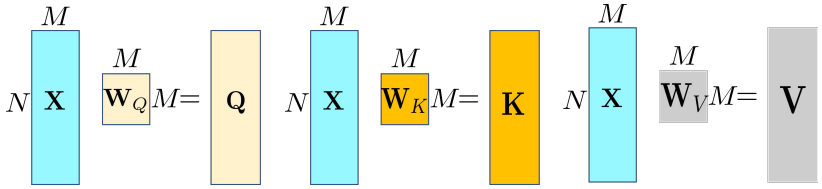

Figure 7: This diagram illustrates the module of Attention. The boxes represent matrices. It depicts the computation process for **Q**, **K**, and **V**.

Here, $\mathbf{x}^i \in \mathbb{R}^{(N,M)}$ represents the input of the i-th layer, while $\mathbf{x}^{i+1} \in \mathbb{R}^{(O,M-k+1))}$ is the output from the same layer. The matrix $\mathbf{W}_i \in \mathbb{R}^{(O,N,K)}$ are convolution kernel with the kernel size is K and

the number of kernel is O. $\mathbf{x}'_{i+1}$, $\mathbf{x}'_i$, $\mathbf{W}'_i$ are transformed from $\mathbf{x}_{i+1} \in \mathbb{R}^{(O(M-k+1),1)}$, $\mathbf{x}_i \in \mathbb{R}^{(NM,1)}$, $\mathbf{W}_i \in \mathbb{R}^{(NM,O(M-k+1)}$ based on Matrix-vector method. The general 1D can be found in section 6

We observe that a single operation of 1D convolution learns the correlations among K columns using the convolution kernel. It generates output for a single column in one row. Sliding the convolution kernel can produce output for multiple columns in the same row, and increasing the number of convolution kernels enables obtaining outputs for multiple columns across multiple rows.

Considering 2D convolution, Figure 5 **a** illustrates the scenario of a 2D convolution with single-channel input and multi-channel output, which can be mathematically expressed as $\mathbf{W}_1 \circledast \mathbf{x} = \mathbf{y}_1 \cdots \mathbf{W}_O \circledast \mathbf{x} = \mathbf{y}_O$. Part **b** of this figure provides a simple example, while part **c** corresponds to the matrix-vector representation of the situation depicted in part **b**. In Figure 6, parts a and b demonstrate the case of a 2D convolution with multi-channel input and multi-channel output, where initially, convolutions are performed on different channels followed by summation. A straightforward illustration is given in part **c**, and its corresponding matrix-vector form is presented in part **d**.

The matrix-vector representation for a 2D convolution with single-channel input and multi-channel output can be written as:

$$[\mathbf{W}_1, \mathbf{W}_2 \cdots \mathbf{W}_O] \diamond \mathbf{x}' = \mathbf{y}' \tag{9}$$

On the other hand, for a 2D convolution with multi-channel input and multi-channel output, the matrix-vector form can be expressed as:

$$\begin{pmatrix} \mathbf{w}'_{1,1} & \mathbf{w}'_{2,1} & \cdots & \mathbf{w}'_{O,1} \\ \mathbf{w}'_{1,2} & \mathbf{w}'_{2,2} & \cdots & \mathbf{w}'_{O,2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{w}'_{1,I} & \mathbf{w}'_{2,I} & \cdots & \mathbf{w}'_{O,I} \end{pmatrix} \diamond \begin{pmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_I \end{pmatrix} = \begin{pmatrix} \mathbf{y}'_1 \\ \mathbf{y}'_2 \\ \vdots \\ \mathbf{y}'_O \end{pmatrix} \tag{10}$$

Given that 3D convolutions are essentially an extrapolation of their 2D counterparts, the matrix-vector formulation for 3D convolutions can be systematically developed by extending the principles used to derive the matrix-vector representation of 2D convolutions. Consequently, it is possible to represent all forms of 1D, 2D, and 3D convolutions using matrix-vector notation, harnessing the power of matrix-vector operations to encapsulate their functionality.

Therefore, the distinction between 2D convolutions compared to 1D convolutions lies in their approach to learning local information and the assumptions they make about global context. Two-dimensional convolutions primarily focus on extracting local features, where the richness of this local feature learning is directly related to the size of the convolutional kernel. When the input consists of multiple channels, 2D convolutions inherently assume that there exists a correlation among the same local positions across different channels, thereby enabling them to learn from multi-channel information simultaneously. Regarding global information, 2D convolutions operate under the assumption that similar local patterns are relevant throughout the entire space. This allows the convolutional kernel to be slid and applied uniformly across the entire global extent, capturing shared local characteristics wherever they may appear.

4.5. Matrix-Vector Method for Transformer

In this chapter, we will employ the Matrix-Vector method to elucidate the inner workings of the Transformer. The Transformer architecture comprises two fundamental components: the Feed-Forward Network (FFN) and the Multi-Head Attention structure. FFNs involve Linear or Convolution operations, which we have already discussed in Sections 4.3 and 4.4. In this chapter,

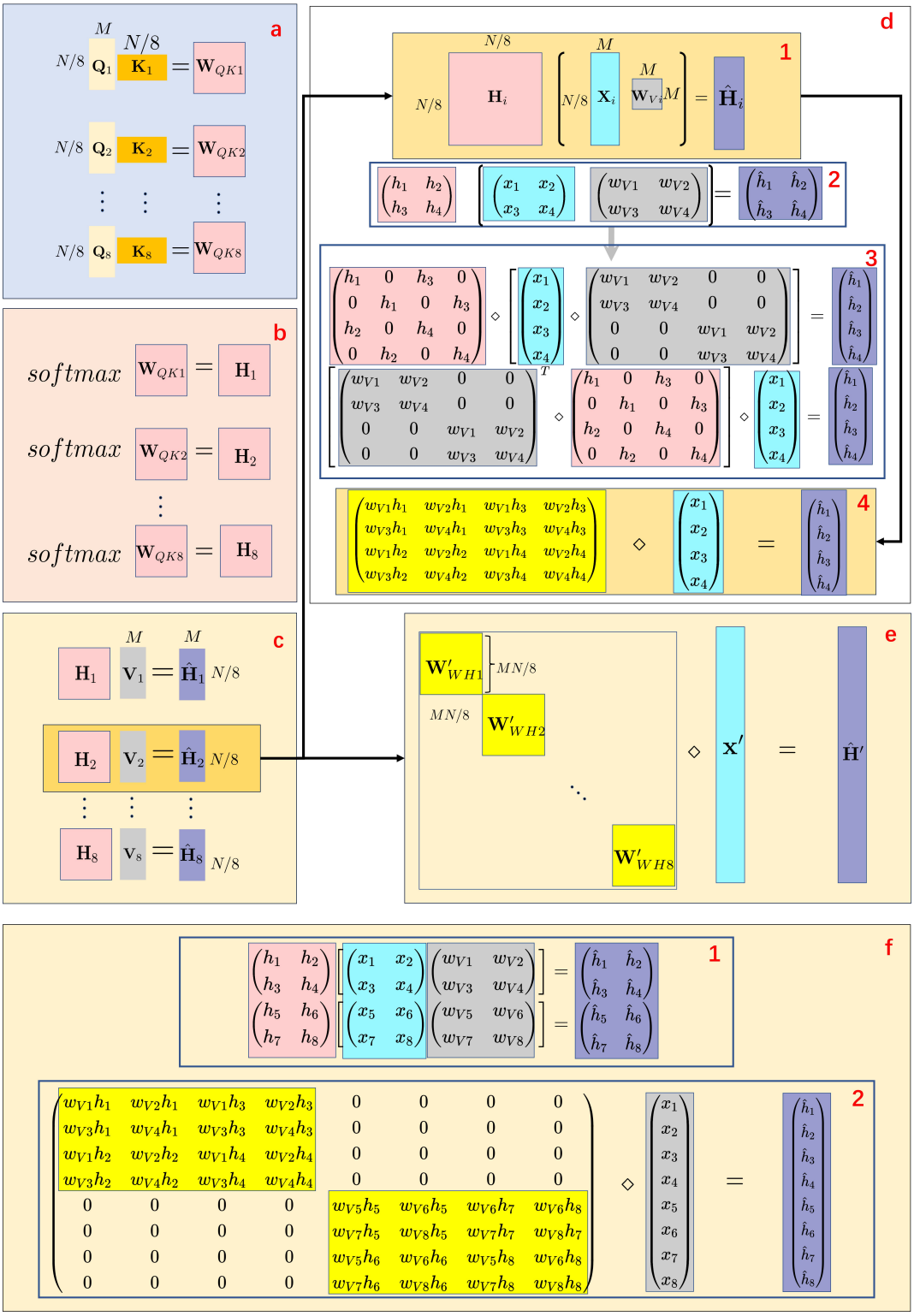

Figure 8: This diagram illustrates the process of transforming Concat$(\hat{\mathbf{H}}_1 ... \hat{\mathbf{H}}_8)$ in the multi-head attention mechanism into its corresponding matrix-vector form $\mathbf{W}'_{WH}\mathbf{x}' = \hat{\mathbf{H}}'$. Different colors in the diagram correspond to different variables.

our primary focus is on the multi-head attention mechanism. Consequently, when we refer to the Transformer henceforth, we are specifically referring to the multi-head attention module within the Transformer. The mechanism is defined by the following equation:

$$\mathring{\mathbf{H}} = \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\left(\hat{\mathbf{H}}_1, \ldots, \hat{\mathbf{H}}_h\right) \mathbf{W}_O \tag{11}$$

where

$$\begin{aligned} \hat{\mathbf{H}}_i &= \text{Attention}\left(\mathbf{x}_i\mathbf{W}_{Qi} = \mathbf{Q}_i, \mathbf{x}_i\mathbf{W}_{Ki} = \mathbf{K}_i, \mathbf{x}_i\mathbf{W}_{Vi} = \mathbf{V}_i\right) \\ &= \text{softmax}\left(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{M}}\right)\mathbf{V}_i \\ &= \mathbf{H}_i\mathbf{V}_i = \mathbf{H}_i[\mathbf{x}_i\mathbf{W}_{Vi}] \end{aligned} \tag{12}$$

Here, h represents the number of attention heads, and the input $\mathbf{x} \in \mathbb{R}^{(N,M)}$ is divided into $\mathbf{x}_1, ..., \mathbf{x}_h$ based on h. The parameters $\mathbf{W}_{Qi}, \mathbf{W}_{Ki}$, and $\mathbf{W}_{Vi}$ correspond to $\mathbf{x}_i$.

Our objective is to express the multi-head attention as $\mathbf{W}'\mathbf{x}'$. Prior to transforming the multi-head attention mechanism into the Matrix-Vector multiplication form, we need to conduct a comprehensive analysis and clearly define the research object. In Equation 12, Concat $\left(\hat{\mathbf{H}}_1, \ldots, \hat{\mathbf{H}}_h\right)$ describes an engineering process that requires mathematical representation. The learning process for the entire input $\mathbf{x}$ primarily consists of two parts: $\mathbf{V}_1, \ldots, \mathbf{V}_h$ and $\mathbf{H}_1, \ldots, \mathbf{H}_h$. The $\mathbf{H}_i$ is derived based on the parameters $\mathbf{W}_{Qi}$ and $\mathbf{W}_{Ki}$ and can be directly regarded as random parameters.

Figure 8 illustrates the process of converting Concat $\left(\hat{\mathbf{H}}_1, \ldots, \hat{\mathbf{H}}_h\right)$ (part c) into the matrix-vector form $\mathbf{W}'_{WH} \diamond \mathbf{x}' = \hat{\mathbf{H}}'$ (part **e**), where $\mathbf{W}'_{WH}$ is generated from $\mathbf{H}_1, \ldots, \mathbf{H}_8$ and $\mathbf{W}_{V1}, \ldots, \mathbf{W}_{V8}$. Part **a** represents the computation process of obtaining $\mathbf{W}_{QKi}$ in the attention mechanism, while part **b** computes $\mathbf{H}_1, \ldots, \mathbf{H}_8$ through the $softmax$ operation. Part **c** computes $\hat{\mathbf{H}}_i[\mathbf{x}_i\mathbf{W}_{Vi}]$. In part d, we provide a simple example demonstrating the conversion of $\hat{\mathbf{H}}_i[\mathbf{x}_i\mathbf{W}_{Vi}]$ into $\mathbf{W}'_{WHi}\mathbf{x}'_i$. This process is divided into four parts: **d.1** represents the general form of $\hat{\mathbf{H}}_i[\mathbf{x}_i\mathbf{W}_{Vi}]$, while **d.2** serves as a simple example of **d.1**. **d.3** first rewrites **d.2** using diamond multiplication as $\hat{\mathbf{H}}'_i \diamond [\mathbf{x}'_i \diamond \mathbf{W}'_{Vi}]$, and then utilizes the property of diamond multiplication to express it as $[\mathbf{W}'^T_{Vi} \diamond \hat{\mathbf{H}}'_i] \diamond \mathbf{x}'_i$. Finally, **d.4** is obtained as $\mathbf{W}'_{WHi}\mathbf{x}'_i$. It can be observed that $\mathbf{W}'_{WHi}$ is a dense matrix. In part **e**, we present the conversion of the entire $\mathbf{H}_1[\mathbf{x}_1\mathbf{W}_{V1}]...\mathbf{H}_8[\mathbf{x}_8\mathbf{W}_{V8}]$ into the matrix-vector form $\mathbf{W}'_{WH}\mathbf{x}'$. In part **f**, we present a simple example of part **e**. **f.1** depicts $\mathbf{H}_i[\mathbf{x}_i\mathbf{W}_{Vi}]$, while **f.2** represents the Matrix-vector format of **f.1**.

Figure 9 illustrates the parameter transformation scenario after incorporating $\mathbf{W}_O$. Part **a** demonstrates $\mathbf{W}_{WH} \diamond \mathbf{x}\mathbf{W}'_O = \mathring{\mathbf{H}}'$, while Part **b** rewrites **a** as $\mathbf{W}^T_{WH}\mathbf{W}'_O \diamond \mathbf{x} = \mathring{\mathbf{H}}'$. Consequently, based on Figures 8 and 9, the entire multi-head attention mechanism can be expressed as $\mathbf{W}'_{HVO} \diamond \mathbf{x}' = \mathring{\mathbf{H}}'$, where $\mathbf{W}'_{HVO} = \mathbf{W}^T_{WH} \diamond \mathbf{W}'_O$. By analyzing the matrix structures of $\mathbf{W}_{WH}$ and $\mathbf{W}'_O$, we can speculate that the addition of $\mathbf{W}'_O$ has minimal impact on the sparsity and shape of $\mathbf{W}_{WH}$. So the whole multi-head attention can be written as:

$$\mathring{\mathbf{H}} = (\mathbf{W}'_{HVO})^T(\mathbf{x}^i)' \tag{13}$$

Here, $\mathbf{x}^i \in \mathbb{R}^{(NM,1)}$ and $\mathring{\mathbf{H}} \in \mathbb{R}^{(NM,1)}$ are the input and output of ith layer, while $\mathbf{W}'_{HVO} \in \mathbb{R}^{(NM,NM)}$ is matrices generated in accordance with $\mathbf{H}$, $\mathbf{W}_V$ and $\mathbf{W}_O$. In this manner, we have expressed the multi-head attention mechanism as a Matrix-Vector multiplication. This matrix multiplication representation provides a more concise way to express the multi-head attention mechanism. The general circumstance deduction is shown in section 6

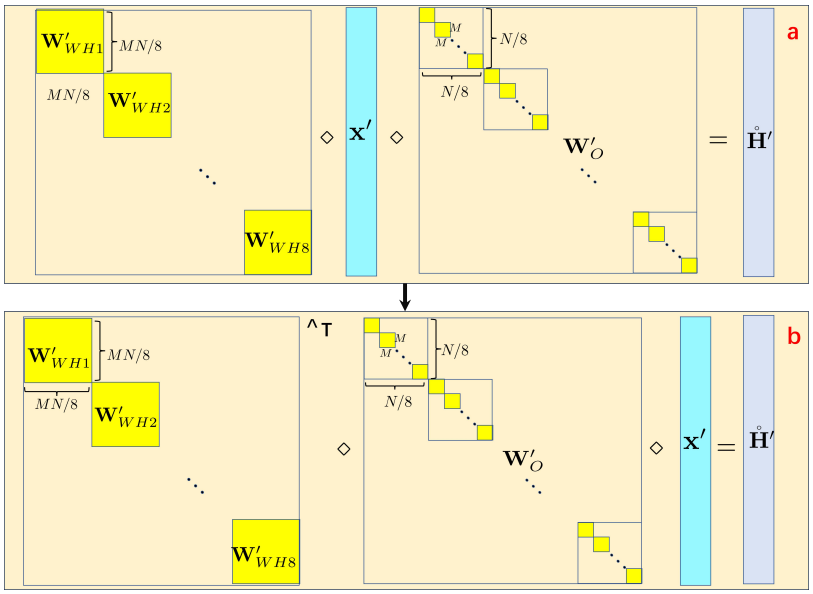

Figure 9: This figure illustrates the process of deriving the matrix-vector form corresponding to the entire multi-head attention mechanism, which is represented as $\mathbf{W}_{WH}^{T} \diamond \mathbf{W}'_{O} \diamond \mathbf{x}'$.

.

Based on the derivations above and referring to Figure 8, we can gain a deeper understanding of the learning process in the multi-head attention mechanism. The multi-head attention can be regarded as a specialized form of local linear transformation, which involves first dividing $\mathbf{x}$ into $\mathbf{x}_1 ... \mathbf{x}_h$ based on the number of heads, then transforming $\mathbf{x}_i$ into $\mathbf{x}'_i$, and subsequently utilizing $\mathbf{H}'_i$, $\mathbf{W}'_{Vi}$, and $\mathbf{W}'_{Oi}$ to generate the corresponding parameters $\mathbf{W}'_{HVOi}$ for $\mathbf{x}'_i$.

4.6. Summary and Sparsity Analysis

In summary, Linear operations, 1D convolutions, 2D convolutions, 3D convolutions, and Transformers can all be represented as matrix-vector multiplications, thereby conforming to the fundamental framework of the Universal Approximation Theory. Consequently, each of multi-layer networks composed by these methodologies is a concrete instantiation of the Universal Approximation Theory. In addition, we give a deduction of the general circumstance of Linear, 1D Convolution, 2D Convolution and Multi-head attention in section 6. We conclude that the sparsity of them are $1/M$, K/M,$K^2/(MN)$,and $1/h$ separately.

5. Discussion

Now we can provide explanations for the previously mentioned questions regarding parameter redundancy, generalization, interpretability, performance bottlenecks, and the future development of deep learning.

Firstly, the existence of parameter redundancy in deep learning is due to the excessively low weights of certain layers. We can analyze this directly from the formula of the universal approximation theorem:

$$|\sum_{j=1}^{N} \alpha_j \sigma\left(\mathbf{W}_j^{\mathrm{T}} \mathbf{x} + \theta_j\right) - f(\mathbf{x})| < \varepsilon \quad \text{for all} \quad \mathbf{x} \in \mathbf{I}_n. \tag{14}$$

Let's assume $N_1 + N_2 = N$ and $|\sum_{j=1}^{N_2} \alpha_j \sigma\left(\mathbf{W}_j^{\mathrm{T}} \mathbf{x} + \theta_j\right)| \to 0$. Then we have:

$$|\sum_{j=1}^{N_1}\alpha_j\sigma\left(\mathbf{W}j^{\mathrm{T}}\mathbf{x}+\theta_j\right)+\sum_{j=1}^{N_2}\alpha_j\sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x}+\theta_j\right)-f(\mathbf{x})|<\varepsilon \tag{15}$$

Since $|\sum_{j=1}^{N_2}\alpha_j\sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x}+\theta_j\right)|\to 0$, we have the following inequality:

$$\begin{aligned}&\left||\sum_{j=1}^{N_1}\alpha_j\sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x}+\theta_j\right)-f(\mathbf{x})|-|\sum_{j=1}^{N_2}\alpha_j\sigma\left(\mathbf{W}j^{\mathrm{T}}\mathbf{x}+\theta_j\right)|\right|\\ \leqslant&|\sum_{j=1}^{N_1}\alpha_j\sigma\left(\mathbf{W}j^{\mathrm{T}}\mathbf{x}+\theta_j\right)+\sum_{j=1}^{N_2}\alpha_j\sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x}+\theta_j\right)-f(\mathbf{x})|<\varepsilon\end{aligned} \tag{16}$$

Therefore, we have:

$$|\sum_{j=1}^{N_1}\alpha_j\sigma\left(\mathbf{W}j^{\mathrm{T}}\mathbf{x}+\theta_j\right)-f(\mathbf{x})|<\varepsilon+|\sum_{j=1}^{N_2}\alpha_j\sigma\left(\mathbf{W}_j^{\mathrm{T}}\mathbf{x}+\theta_j\right)| \tag{17}$$

Hence, when parameters in certain layers are small enough, we can directly remove those layers since their impact on the final result is minimal.

The second question regarding the generalization capability of deep learning is related to the approximation conditions in the universal approximation theorem. Current evidence shows that this theorem can approximate any Borel measurable function, which typically encompasses step functions, continuous functions, piecewise continuous functions, and indicator functions. Hence, when there exist multiple continuous functions within a closed interval, it is inherently challenging for a network to precisely approximate all these functions simultaneously. This inherent limitation forms the fundamental reason why deep learning encounters performance bottlenecks on certain publicly available datasets.

Regarding interpretability in deep learning, it is unnecessary to rely solely on visualization techniques for intermediate layers. The universal approximation theorem can be understood as solving a system of functional equations; we are essentially searching for the set of parameters corresponding to the solutions of these equations. Increasing the number of network layers and applying operations like multi-scale processing effectively enhance the value of N in the context of the universal approximation theorem. Overfitting and underfitting represent errors in the process of function approximation, and a large volume of data enables the network to approximate higher-dimensional functions that better fit most of the data. Thus, one direction for future efforts should involve directly solving for the corresponding parameters based on the data, rather than relying solely on forward and backward propagation.

In terms of the future development of deep learning, greater attention should indeed be given to the data itself, particularly Borel measurable functions. Since the universal approximation theorem demonstrates its ability to approximate any such function within a closed interval, an approach could involve classifying the data and dividing it into categories according to the Borel measurable functions within those intervals, using distinct networks to fit each category. Alternatively, algorithms could be designed to transform all data into Borel measurable functions.

Furthermore, this paper argues that since the vast majority of deep learning models embody the principles of the universal approximation theorem, there is less need to continually design copious new models. Instead, emphasis should be placed on leveraging pre-trained models effectively, as retraining from scratch each time can be computationally resource-intensive. Consequently, the endeavor to repurpose well-trained models in novel tasks holds significant practical value. The

theoretical groundwork for addressing this issue lies in the adjustment of approximating functions tailored to new problems.

We propose an approach that involves freezing the original layer functions and subsequently adding new fine-tuning layers atop them, as depicted in Figure 10, where the orange layers represent the original ones, and the blue layers denote the newly appended ones. The rationale behind fixing the original parameters is that they are typically trained on vast amounts of data, thereby possessing a degree of generalizability; while the added layers allow for fine-tuning the entire approximation model.

Equation 18 presents a generic representation of a two-layer network, while Equation 19 illustrates the corresponding approximation form after incorporating the fine-tuning layers (in both Equations 18 and 19, we have directly represented the models in matrix-vector notation for simplicity). In essence, our method maintains the integrity of the pre-trained layers while introducing adaptability through additional trainable layers, enabling the model to effectively learn from new or specialized data without discarding the valuable knowledge encoded in the original parameters.

$$\begin{aligned} \mathbf{x}'_{i+1} = B_i(\mathbf{x}'_i) &= \mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) \\ \mathbf{x}'_{i+2} = B_{i+1}(\mathbf{x}'_{i+1}) &= \mathbf{W}'_{(i+1)1}B_i(\mathbf{x}'_i) + \sigma(\mathbf{W}'_{(i+1)2}B_i(\mathbf{x}'_i)) \\ &= \mathbf{W}'_{(i+1)1}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i)] + \sigma(\mathbf{W}'_{(i+1)2}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i)]) \end{aligned} \tag{18}$$

$$\begin{aligned} \mathbf{x}'_{(i+1)f} &= B_{if}(\mathbf{x}'_i) = \sigma(\mathbf{W}'_{if}\mathbf{x}'_i) \\ \mathbf{x}'_{i+1} = \mathbf{x}'_{i+1} + \mathbf{x}'_{(i+1)f} &= \mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i) \\ \mathbf{x}'_{i+2} = \mathbf{x}'_{i+2} + \mathbf{x}'_{(i+2)f} &= \mathbf{W}'_{(i+1)1}\mathbf{x}'_{i+1} + \sigma(\mathbf{W}'_{(i+1)2}\mathbf{x}'_{i+1}) + \sigma(\mathbf{W}'_{(i+1)f}\mathbf{x}'_{i+1}) \\ &= \mathbf{W}'_{(i+1)1}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i)] \\ &\quad + \sigma(\mathbf{W}'_{(i+1)2}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i)]) \\ &\quad + \sigma(\mathbf{W}'_{(i+1)f}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i)]) \end{aligned} \tag{19}$$

It is evident that we have made targeted adjustments to $\sigma(\mathbf{W}'_{i2}\mathbf{x}'_i)$ in $\mathbf{x}'_{i+1}$ and $\sigma(\mathbf{W}'_{(i+1)2}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i)])$ in $\mathbf{x}'_{i+2}$, thereby enabling the model to effectively approximate corresponding functions within new problem contexts. Given the Universal Approximation Theorem's reliance on the collective approximation by multiple nonlinear functions (such as $\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$), it is feasible in practice to optimize only a subset of layers to adapt to novel data, significantly conserving computational resources.

While this strategy is theoretically sound and partially realized, specific conditions are imposed on parameters before and after adjustment. Ideally, the fine-tuned parameters, $\sigma(\mathbf{W}'_{if}\mathbf{x}'_i)$ and $\sigma(\mathbf{W}'_{(i+1)f}[\mathbf{W}'_{i1}\mathbf{x}'_i + \sigma(\mathbf{W}'_{i2}\mathbf{x}'_i) + \sigma(\mathbf{W}'_{if}\mathbf{x}'_i)])$, should either approach zero or require minimal tweaking of select layers. The rationale for reusing pre-trained models lies in their generalizability, typically trained on large-scale datasets, while new data often pertains to specific domains. If the disparity between the original model and the new domain-specific dataset is substantial, then fine-tuning may essentially amount to training a new model from scratch, defeating the purpose of saving computational resources and potentially wasting them through loading the original model. However, the challenge arises in the inability to anticipate the extent of difference between the original and fine-tuned models.

In response to this issue, this paper suggests a reference solution: initially train a model F using a portion of the original dataset, followed by calculating the differences between the parameters of this trained model F and those of the original model G. If the discrepancy is relatively minor or

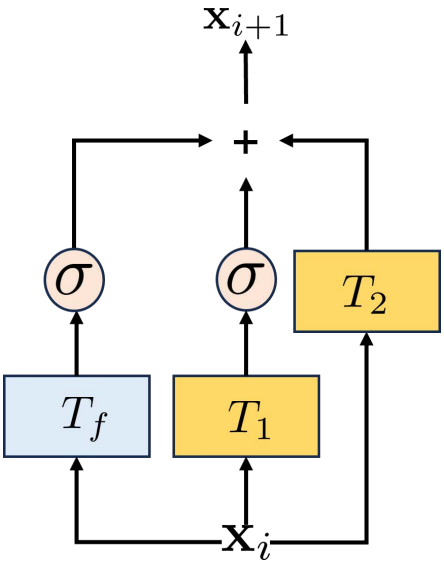

Figure 10: The general method of fine-tuning a ResNet block. We will fix parameters in the orange boxes and fine-tune the parameters in the blue box.

confined to a limited number of layers, this indicates that the original model G is indeed suitable for fine-tuning to the new task; otherwise, fine-tuning might not be the optimal approach.

Essentially, this method advocates adaptive and targeted fine-tuning of existing models instead of reinventing the wheel, which not only economizes computational resources but also potentially accelerates progress across diverse problem domains.

6. Conclusion

In this paper, we delve into the foundational issues of deep learning theory. Specifically, I introduce the Matrix-Vector Method, a technique that unifies Linear, Transformer, and Convolutional modules under a common mathematical framework. Leveraging this method, the current deep learning models can be unified within the framework of the Universal Approximation Theory. Integrating the Universal Approximation Theory offers a solution to some of the existing challenges in current deep learning.

Comments

This article represents the culmination of five years of work for me, completed during my doctoral application period. It embodies what I aspired to achieve since the beginning of my graduate studies in 2018. At that time, I genuinely believed that deep learning was a magical and essential element for the future of humanity. However, when I attempted to implement my own models, they didn't perform as well as others', leaving me consistently perplexed. At times, I felt that deep learning was akin to building with blocks, and I envied those who published numerous papers. So, I set out to uncover the fundamental principles of deep learning and achieve success step by step.

Subsequently, I delved into researching and exploring the relationships among deep learning parameters. I also reviewed existing theoretical or explanatory papers on deep learning, but I found them lacking compared to my research (a belief I still hold today). Their studies quickly become obsolete, whereas what I did 3-4 years ago remains relevant and aligns with the principles of

universal approximation theory. However, when I revisit the articles I wrote earlier, I feel nauseated; they are poorly written.

Amidst my confusion, I persisted in learning. Finally, one day, seemingly in my third year of graduate studies, I had a revelation that the fundamental solution to deep learning lies in matrix multiplication. This idea lingered, and I proceeded to provide some proofs. The more I proved, the more convinced I became. Although some of the proofs may have flaws in hindsight, the overall direction was correct. Later, I introduced the matrix-vector method, unifying everything under the framework of the universal approximation theorem. I knew I was on the right track.

However, this article remains unpublished. Perhaps it's due to my poor narrative skills. When I read articles published by others, I am genuinely perplexed. I feel lost and unfair. Some of what they do seems essentially worthless. They may not even use the theories they propose. Reviewing my previous research, it essentially adheres to the fundamental principles of deep learning: the universal approximation theory, but it still cannot be published. I do not understand the significance of academic research. Is a unified deep learning article like mine not worth publishing? I don't know if it will ever be recognized and brought up by other scholars. Perhaps, my paper will fade into the river of time, producing no ripples.

The Property of Diamond

There are tow properties are always used: $\mathbf{W}_1 \diamond [\mathbf{x} \diamond \mathbf{W}_2] = \mathbf{W}_2^T \diamond \mathbf{W}_1 \diamond \mathbf{x} = \mathbf{W}_{12}$ and $[\mathbf{W}_1 \diamond \mathbf{x}] \diamond \mathbf{W}_2 = \mathbf{W}_1^T \diamond \mathbf{W}_2 \diamond \mathbf{x}$, where$\mathbf{W}_{12} = \mathbf{W}_1^T \mathbf{W}_2^T$. They are proved in Eq..1 and .2.

$$
\begin{aligned}
&\mathbf{W}_1 \diamond [\mathbf{x} \diamond \mathbf{W}_2] \\
=&\mathbf{W}_1 \diamond [\mathbf{W}_2^T \mathbf{x}] \\
=&\mathbf{W}_1^T [\mathbf{W}_2^T \mathbf{x}] \\
=&\mathbf{W}_1^T \mathbf{W}_2^T \mathbf{x} = \mathbf{W}_{12} \mathbf{x} \\
=&\mathbf{W}_2^T \diamond \mathbf{W}_1 \diamond \mathbf{x}
\end{aligned}
\tag{.1}
$$

$$
\begin{aligned}
&[\mathbf{W}_1 \diamond \mathbf{x}] \diamond \mathbf{W}_2 \\
=&[\mathbf{W}_1^T \mathbf{x}] \diamond \mathbf{W}_2 \\
=&\mathbf{W}_2^T [\mathbf{W}_1^T \mathbf{x}] \\
=&\mathbf{W}_1^T \diamond \mathbf{W}_2 \diamond \mathbf{x}
\end{aligned}
\tag{.2}
$$

The Sparsity and Matrix-Vector Form

In this section, we will present the generalized representations of Linear, Convolution, and Transformer transformed into their corresponding matrix-vector forms, and analyze the complexity of the corresponding matrices.

The Matrix-Vector Form of Linear

According to Eq..3, it can be deduced that if the input data matrix has dimensions (N, M) and the parameter matrix has dimensions (N, N), the sparsity of the parameter matrix corresponding to the Linear operation is $1/M$.

The Matrix-Vector Form of Convolution

Eq. .4 illustrates the transformation of a single-channel output convolution into the Matrix-Vector form, while Eq. .5 extends this representation to a multi-channel scenario. From Eq. .4 and Eq. .5, it can be deduced that when the input data matrix has dimensions (N, M) and the convolution kernel has dimensions (K, K), where $K \ll M$, the sparsity of the parameter matrix corresponding to the 1D convolution operation is K/M.

The Sparsity of Multi-head Attention

The multi-head attention mechanism is relatively complex, making it difficult to directly represent it in matrix-vector form. Therefore, we choose to first compute the sparsity of a single attention mechanism. The calculation process is shown in Eq..6. Due to its complexity, we only provide the output of the first node in Eq..7. According to Eq..7, we can infer that the parameter matrix corresponding to the attention mechanism is a dense matrix. If both matrices on the left and right sides do not contain zeros, it corresponds to a matrix with no 0 elements, and its sparsity is 0.

Combining Figures 8 and 9, we can speculate that the addition of $\mathbf{W}'_O$ hardly alters the sparsity of the attention mechanism's $\mathbf{W}_{WH}$. Consequently, we can infer that the sparsity of the parameter matrix corresponding to the multi-head attention mechanism is $1/h$.

$$\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n,1} & y_{n,2} & \cdots & y_{n,m} \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} w_{1,1} & 0 & \cdots \\ 0 & w_{1,1} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ w_{1,2} & 0 & \cdots \\ 0 & w_{1,2} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \cdots \\ w_{1,n} & 0 & \cdots \\ 0 & w_{1,n} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \end{pmatrix} \diamond \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,m} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,m} \\ \vdots \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ \vdots \\ y_{1,m} \\ y_{2,1} \\ y_{2,2} \\ \vdots \\ y_{2,m} \\ \vdots \\ y_{n,1} \\ y_{n,2} \\ \vdots \\ y_{n,m} \end{pmatrix} \tag{.3}$$

$$\begin{aligned} & w_{1,1}(x_{1,1}w'_{1,1} + x_{1,2}w'_{2,1} + x_{1,m}w'_{m,1}) \\ & +w_{1,2}(x_{2,1}w'_{1,1} + x_{2,2}w'_{2,1} + x_{2,m}w'_{m,1}) \\ & + \cdots \\ & +w_{1,n}(x_{n,1}w'_{1,1} + x_{n,2}w'_{2,1} + x_{n,m}w'_{m,1}) \end{aligned} \tag{.7}$$

References

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, p. 303–314, Jan 2007. [Online]. Available: http://dx.doi.org/10.1007/bf02551274

[2] K. Hornik, M. B. Stinchcombe, and H. L. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989. [Online]. Available: https://api.semanticscholar.org/CorpusID:2757547

$$\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,k} \end{pmatrix} \circledast \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots & & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} & \cdots & x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m-k+1} \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} w_{1,1} & 0 & \cdots \\ w_{1,2} & w_{1,1} & \cdots \\ \vdots & \vdots & \cdots \\ w_{1,k} & w_{1,k-1} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ w_{2,1} & 0 & \cdots \\ w_{2,2} & w_{2,1} & \cdots \\ \vdots & \vdots & \cdots \\ w_{2,k} & w_{2,k-1} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \cdots \\ w_{n,1} & 0 & \cdots \\ w_{n,2} & w_{n,1} & \cdots \\ \vdots & \vdots & \cdots \\ w_{n,k} & w_{n,k-1} & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \end{pmatrix} \diamond \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ w_{1,k} \\ \vdots \\ x_{1,m} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ w_{2,k} \\ \vdots \\ x_{2,m} \\ \vdots \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{k,k} \\ \vdots \\ x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ \vdots \\ y_{1,m-k+1} \end{pmatrix} \tag{.4}$$

$$\begin{pmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,k} \\ w^1_{2,1} & w^1_{2,2} & \cdots & w^1_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w^1_{n,1} & w^1_{n,2} & \cdots & w^1_{n,k} \end{pmatrix} \circledast \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots & & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} & \cdots & x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m-k+1} \end{pmatrix}$$

$$\begin{pmatrix} w^2_{1,1} & w^2_{1,2} & \cdots & w^2_{1,k} \\ w^2_{2,1} & w^2_{2,2} & \cdots & w^2_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w^2_{n,1} & w^2_{n,2} & \cdots & w^2_{n,k} \end{pmatrix} \circledast \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots & & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} & \cdots & x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{2,1} & y_{2,2} & \cdots & y_{2,m-k+1} \end{pmatrix}$$

$\cdots\cdots$

$$\begin{pmatrix} w^O_{1,1} & w^O_{1,2} & \cdots & w^O_{1,k} \\ w^O_{2,1} & w^O_{2,2} & \cdots & w^O_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w^O_{n,1} & w^O_{n,2} & \cdots & w^O_{n,k} \end{pmatrix} \circledast \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots & & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} & \cdots & x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{O,1} & y_{O,2} & \cdots & y_{O,m-k+1} \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} w^1_{1,1} & 0 & \cdots & w^2_{1,1} & 0 & \cdots & w^O_{1,1} & 0 & \cdots \\ w^1_{1,2} & w^1_{1,1} & \cdots & w^2_{1,2} & w^2_{1,1} & \cdots & w^O_{1,2} & w^O_{1,1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ w^1_{1,k} & w^1_{1,k-1} & \cdots & w^2_{1,k} & w^2_{1,k-1} & \cdots & w^O_{1,k} & w^O_{1,k-1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\ w^1_{2,1} & 0 & \cdots & w^2_{2,1} & 0 & \cdots & w^O_{2,1} & 0 & \cdots \\ w^1_{2,2} & w^1_{2,1} & \cdots & w^2_{2,2} & w^2_{2,1} & \cdots & w^O_{2,2} & w^O_{2,1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ w^1_{2,k} & w^1_{2,k-1} & \cdots & w^2_{2,k} & w^2_{2,k-1} & \cdots & w^O_{2,k} & w^O_{2,k-1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ w^1_{n,1} & 0 & \cdots & w^2_{n,1} & 0 & \cdots & w^O_{n,1} & 0 & \cdots \\ w^1_{n,2} & w^1_{n,1} & \cdots & w^2_{n,2} & w^2_{n,1} & \cdots & w^O_{n,2} & w^O_{n,1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ w^1_{n,k} & w^1_{n,k-1} & \cdots & w^2_{n,k} & w^2_{n,k-1} & \cdots & w^O_{n,k} & w^O_{n,k-1} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \end{pmatrix} \diamond \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ w_{1,k} \\ \vdots \\ x_{1,m} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ w_{2,k} \\ \vdots \\ x_{2,m} \\ \vdots \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{k,k} \\ \vdots \\ x_{n,m} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ \vdots \\ y_{1,m-k+1} \\ y_{2,1} \\ y_{2,2} \\ \vdots \\ y_{2,m-k+1} \\ \vdots \\ y_{O,1} \\ y_{O,2} \\ \vdots \\ y_{O,m-k+1} \end{pmatrix} \tag{.5}$$

$$\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{n,n} \end{pmatrix} \left[\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \begin{pmatrix} w'_{1,1} & w'_{1,2} & w'_{1,3} & \cdots & w'_{1,m} \\ w'_{2,1} & w'_{2,2} & w'_{2,3} & \cdots & w'_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ w'_{m,1} & w'_{m,2} & w'_{m,3} & \cdots & w'_{m,m} \end{pmatrix} \right]$$

$$= \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{n,n} \end{pmatrix}$$

$$\begin{pmatrix} x_{1,1}w'_{1,1} + x_{1,2}w'_{2,1} + x_{1,m}w'_{m,1} & x_{1,1}w'_{1,2} + x_{1,2}w'_{2,2} + x_{1,m}w'_{m,2} & \cdots & x_{1,1}w'_{1,m} + x_{1,2}w'_{2,m} + x_{1,m}w'_{m,m} \\ x_{2,1}w'_{1,1} + x_{2,2}w'_{2,1} + x_{2,m}w'_{m,1} & x_{2,1}w'_{1,2} + x_{2,2}w'_{2,2} + x_{2,m}w'_{m,2} & \cdots & x_{2,1}w'_{1,m} + x_{2,2}w'_{2,m} + x_{2,m}w'_{m,m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1}w'_{1,1} + x_{n,2}w'_{2,1} + x_{n,m}w'_{m,1} & x_{n,1}w'_{1,2} + x_{n,2}w'_{2,2} + x_{n,m}w'_{m,2} & \cdots & x_{n,1}w'_{1,m} + x_{n,2}w'_{2,m} + x_{n,m}w'_{m,m} \end{pmatrix} \tag{.6}$$
