PHYSOR 2020: Transition to a Scalable Nuclear Future
Cambridge, United Kingdom, March 29th-April 2nd, 2020

SURROGATE MODEL OPTIMIZATION OF A 'MICRO CORE' PWR FUEL ASSEMBLY ARRANGEMENT USING DEEP LEARNING MODELS

Andy Whyte[1] **and Geoff Parks**[1]
[1]University of Cambridge
Department of Engineering, Trumpington Street, Cambridge, CB2 1PZ

ajw287@cam.ac.uk, gtp10@cam.ac.uk,

ABSTRACT

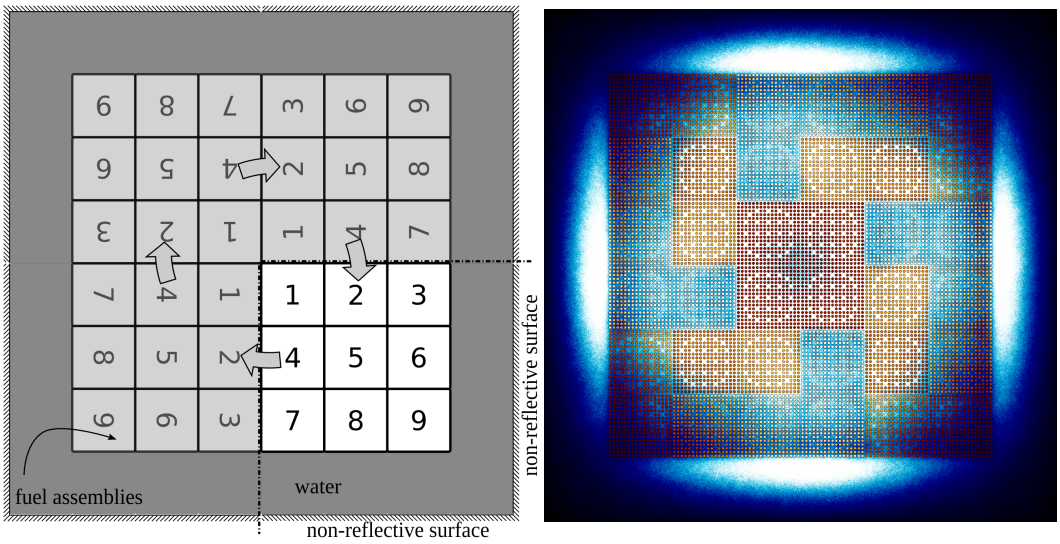
This paper investigates the applicability of surrogate model optimization (SMO) using deep learning regression models to automatically embed knowledge about the objective function into the optimization process. This paper demonstrates two deep learning SMO methods for calculating simple neutronics parameters. Using these models, SMO returns results comparable with those from the early stages of direct iterative optimization. However, for this study, the cost of creating the training set outweighs the benefits of the surrogate models.

KEYWORDS: deep learning, fuel management, PWR, optimization, surrogate model

1. INTRODUCTION

This paper explores two deep learning regression models used with iterative optimization to create a SMO process.The surrogate models are evaluated in the exercise of optimizing the design of a 'micro core' simulation, which is constructed from 36 'standard' PWR fuel assemblies. The design is considered with order four rotational symmetry, reducing the problem to nine assemblies (Fig. 1a). Although there are legitimate limitations of this core design for applications in the real world, it is ideal for investigation and optimization strategy evaluation. The designs are relatively easy to simulate, the design space is bounded, and the design can also be optimized by human experts.

Deep Multi Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) are used as surrogate models that predict the core parameters. The parameters to be optimized were the power peaking factor (PPF) and the position of the hottest pin at the beginning of cycle (BOC); these are outputs in the MLP model and derived from pin power predictions in the CNN. The networks were trained on a set of up to 3200 uniform randomly generated core designs and predictive performance was evaluated using another 400 similarly random designs. The software design philosophy adopted was to use standardised state-of-the-art implementations of recognised software techniques, and open source libraries where possible. In particular, recent years have seen the release of the advanced machine learning libraries tensorflow [2] and Keras [3] and the optimization library pygmo2 [4].



(a) Layout of fuel assemblies in the optimization problem

(b) Flux distribution from an example core simulation

Figure 1: The 'micro core' used in this study [1]

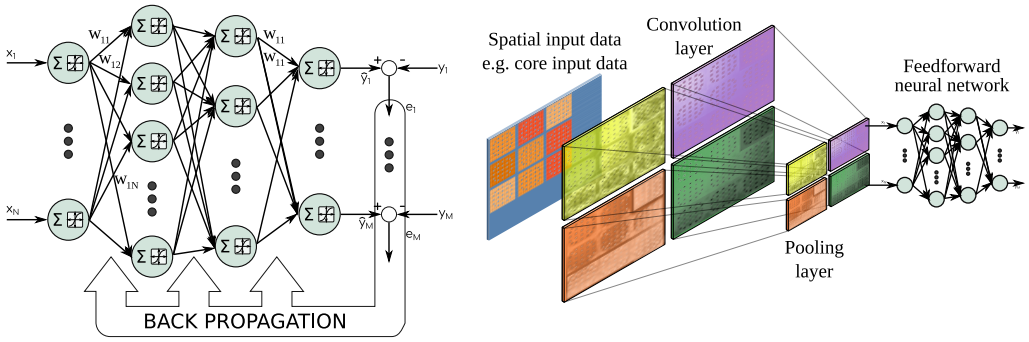
The deep learning models that are derived are then used with a benchmark iterative optimization algorithm: the Non-dominated Sorting Genetic Algorithm (NSGA2) [5]. The speed of the deep learning models enable them to evaluate many thousands of solutions per second. This means that the optimization algorithm could be given larger populations for the same computational budget or that more exploratory settings can be chosen, while using only a fraction of the computational time that a more traditional direct design evaluation approach would require. Results are compared for performance in comparison using the same set-up of optimization algorithm in this study.

2. SURROGATE MODEL OPTIMIZATION

SMO is becoming popular in the field of nuclear engineering [6]. The method is advised for problems where the objective function is computationally expensive [7][8] and works by applying a standard optimization algorithm on a 'surrogate function', which is usually regressed from data points obtained by sampling the original objective function. For simplicity, the SMO methods used the same settings as the direct optimization. To some extent this negates a major attraction of using SMO, namely that more searching can be done on the surrogate model than could be done on the original function for a given computational budget.

2.1. Deep Learning Models

The first type of neural network model used in this paper is an MLP type, a simple feedforward neural network commonly used for regression and classification tasks. As shown in Fig. 2a: an array of processing nodes connect inputs, x, to outputs, y; each node sums the inputs and applies a transform to the data; and between nodes a weighting factor $w_{i,j}$ is applied. The weights, w, define an approximation of the required outputs, y. The backpropagation algorithm made popular by Rummelhart et al. [9] is used to optimize the weights of the network.



(a) A simplified diagram of an MLP **(b) A representation of the CNN layer structure**

Figure 2: Deep learning topologies [1]

The second type of surrogate model is a CNN [10], represented diagrammatically in Fig. 2b. This configuration of network is well suited to spatial data, such as recognition of images [11]. In this architecture, a population of filter kernels perform 'convolutions' on the input image. The resulting images have a sensitivity to the pattern of the kernel that gave rise to them. These images are usually put through a 'pooling layer' (scaled down) and these layers are repeated a number of times. The network finishes up with a deep feedforward neural network similar to an MLP. The system is successful because it does not require the designer to specify the base 'kernels' that will perform low-level image processing; they arise naturally from the convolution layers of the CNN.

The CNN directly predicts the per-pin powers. The objective variables are then calculated from the surrogate outputs by the usual means. These values were chosen based on comparison with existing example networks from literature [12][13][14] and experimentation to fit the input image size and output data. This type of network has proven extremely effective at recognising images, due to its translational invariance. Although translational invariance does not automatically occur in nuclear core design, by selecting inputs to represent the water gap and control/instrumentation pins, the effects of geometric variance are encoded into the inputs of the networks. The CNN represents an example of a typical advanced deep learning tool.

3. MICRO CORE OPTIMIZATION

Experiments are carried out on a small 'toy' example of a fuel arrangement problem, as described above, to explore the use of a SMO approach to fuel management problems. The 'micro core' arrangement of 36 fuel assemblies is to be optimized. For simulation, nine assemblies are arranged into a square lattice, quadrant rotational symmetry is applied on two sides, and they are surrounded by water at an equal thickness to the assemblies followed by non-reflective (black) boundaries (Fig. 1a). The assemblies, labelled 1–9, are standard PWR type assemblies separated by a small water gap. Each assembly has an enrichment value that can be varied independently with quantised enrichment values at increments of 0.2 between 0.8 and 5.0 w/o U235. Monte Carlo simulations were run for uniformly random uranium enrichments in each of the nine assemblies; these form an initial set of training simulations used to train surrogate models. In this study, all 'direct' Monte Carlo neutronics simulations were carried out using the Serpent software [15].

Two output variables are optimized: PPF and the position of the hottest pin. PPF measures the

uniformity of the power generated [16, p73]. The system operates more efficiently if the PPF is lower [17, p374]. By simultaneously moving the hottest pin to the outside of the core and reducing the PPF, a flatter power profile is achieved, which results in a hotter overall coolant outlet temperature.

3.1. Optimization Algorithm

NSGA2 is considered a 'solid multi-objective algorithm'. It is widely used in many real-world applications and is easily parallelised [18]. The parameter values used in this study are shown in Table 1. Note that the number of generations, g, and population size, P, are much smaller than used in other studies, such as [5] ($P = 100$), [19] ($g = 51, P = 204$) and [20] ($P = 300$). This was considered necessary to ensure that the direct optimization could actually be carried out despite its very high computational cost. Although NSGA2 is no longer considered cutting edge, it is still widely used as a benchmark algorithm when comparing novel algorithms or techniques. The implementation used in these experiments is from the software library pygmo [4].

Table 1: Parameters used in NSGA2

Parameter	Value
Population size, P	60
Generations, g	50
Crossover probability	0.95
Distribution index for crossover, η_c	10
Mutation probability	0.01
Distribution index for mutation, η_m	50

4. SURROGATE MODEL EXPERIMENTS

4.1. MLP

A simple deep MLP is used to predict the PPF and horizontal and vertical positions of the hottest pin. Using the w/o U235 on a per-assembly level as the inputs, this means that the input dimension for a quarter core is nine, while the output dimension is three (the hottest pin coordinates and PPF). In order to select an appropriate topology for the MLP, a short study of network topology was carried out. The numbers of hidden layers and neurons per hidden layer were varied. Figure 3 shows the results. The choice to keep the number of neurons per layer constant was made in order to reduce complexity and keep the design tractable on a two-dimensional heatmap. The performance seen in Fig. 3 confirms that MLP networks are robust to a wide variety of topological values. This has been discussed in the literature by many authors (see [21, p130] and more recent deep learning discussion starts with [22]). A fairly large network with nine layers of 80 neurons was chosen for these experiments. The weights are modified using backpropagation and the 'Adam' algorithm [23] and the loss function used was mean absolute error (MAE). The network was trained for 500 epochs. In each training epoch, the system trained on a set of 50 randomly selected samples from the training set, N. This helps to reduce over-fitting of the network to the training set compared to training on the whole set.

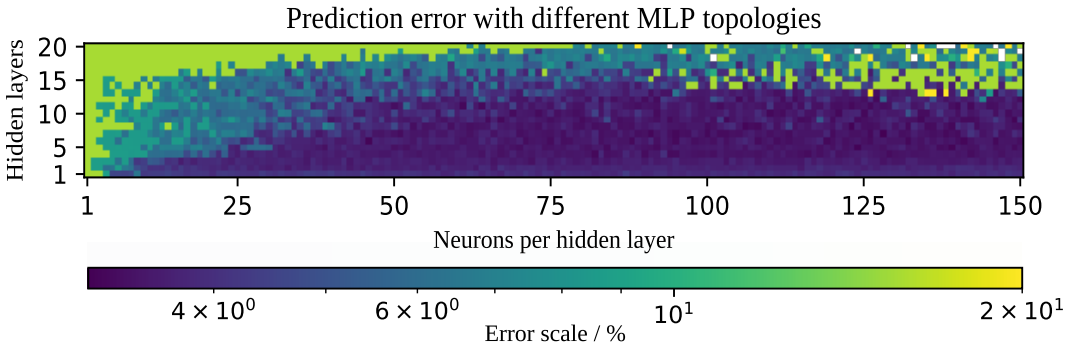


Figure 3: Heatmap of MLP topology (`arch_heatmap_flat.py`) [1]

4.2. CNN

The topology of the CNN used is shown in Table 3 in APPENDIX A. Three convolution and pooling layers are followed by six layers of fully connected feedforward layers. As with the MLP, 'Adam' is used to modify weights based on a MAE loss function. The training was carried out for 100 epochs with a sample size of 50. Unlike in the MLP surrogate model, the CNN is used to predict pin-by-pin power of the system.

5. RESULTS AND DISCUSSION

Simulations show that the surrogate models achieve moderate errors for the objective variables on a test set of random data and are able to generate results using hundreds of thousands of times less computational resource. The CNN predicted pin powers with a MAE around 1%; however, this translated to MAE values of 3.805% and 2.421% for the actual objective variables over the test set. Execution time measurements in Table 2 are shown relative to MLP training time in order to compensate for effects specific to the hardware and software platform. For reference, the mean MLP training time was 172.4 seconds per CPU over 30 iterations on the low-end hardware used.

NSGA2 was used with the MLP and CNN surrogate models and in a direct iterative optimization using Serpent for solution evaluation. The Pareto fronts for the populations of SMO processes are then re-evaluated using Serpent (Fig. 4). The SMO outputs are compared with the Pareto fronts from populations of different generations in the direct optimization. It can be seen that the MLP SMO process yields a Pareto front that has similar performance to the Pareto front of the direct optimization for generation five, while the CNN SMO process has a Pareto front that has similar performance to the twelfth generation. When the NSGA2 runs using a surrogate model evaluator, the computational resource usage is less than one millionth that of the direct optimization process.

The MLP outperforms the CNN at predicting PPF but underperforms at predicting the hot pin location. The CNN performs better when predicting non-random (optimization) input data. This is believed to be because the CNN is predicting a more fundamental system parameter and is more robust to the extrapolation that occurs when the optimization search moves the input space.

The surrogate models investigated here have the potential to reduce computational resource by up to 25%, assuming that the training set already exists. However, the training set generation is not justified in terms of computational expense. If data created during a design study already exists,

Table 2: MAE for MLP, CNN and Monte Carlo on training data (mean values for 30 runs)

	MLP	CNN	Monte Carlo
Training set, N	3200	3200	
Test set	800	800	
MAE per-pin /%	-	0.9949	29.31E-5
error for PPF/%	2.374	3.805	29.31E-5
error for hot pin /%	4.051	2.421	-
relative CPU time (evaluation)	10^{-7}	10^{-5}	$\sim 18^{a}$
relative CPU time (training)	1.000	15.26	-
relative CPU time (NSGA2)	0.03	0.4	$\sim 10^{7a}$

[a] HPC nodes used for Monte Carlo (specifications available at [24])

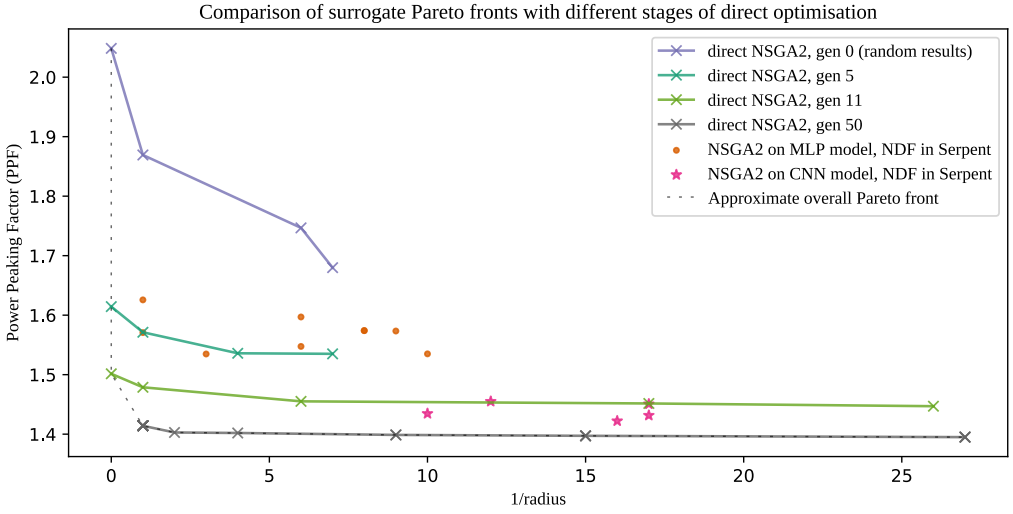


Figure 4: Pareto fronts for populations in direct iterative optimization vs MLP and CNN surrogate models (N=3200, `graph_results_7.py`)

such as in previous work by the authors [25], or if an optimization is going to be carried out on a regular basis, then creation of a surrogate can be justified. Random training data is equivalent to the random initial populations used by many iterative optimization algorithms (including NSGA2) and might be reused to create a surrogate model. The CNN surrogate model significantly outperforms the MLP model for SMO in this case study. If the optimization algorithm used a population of 300, as per [20], and the CNN surrogate still performed comparably for twelve generations, then a net computational saving would be seen. This is not inconceivable since population size primarily discourages premature convergence to local optima rather than changing the rate of Pareto front progression [26].

Further work should establish the investment versus benefit of SMO with regards to direct optimization where population size is similar to that in other studies using NSGA2 (see section 3.1). This could make a case for creating a surrogate model. As the computational cost of the objective function increases, the justification for SMO increases, so larger problems will be more easily justified. Another possibility is to consider intelligent sampling method for training data (e.g. latin hypercube [27] or Sobol sampling [28]). By training the surrogate model on a more space-filling training set, the surrogate model might compete more effectively during optimization. Smaller

training sets could also be considered to justify the surrogate model.

Access to source code for reproducibility and derived works via:
https://github.com/ajw287/microcore-surrogate-physor

6. CONCLUSIONS

In this study two deep learning surrogate models have been applied to optimization of a simple BOC loading pattern problem. The results show that surrogate models can accelerate optimization at the start of the process. This has immediate applications when a corpus of training data has already been produce; data created during a design study or as a by-product of traditional iterative optimizations could be repurposed. The requirement of a large training set offsets the gains of the surrogate model in this study. However, a number of interesting avenues of further research exist, which might tip the computational efficiency balance.

REFERENCES

[1] "Figures for Physor 2020." Zenodo: DOI10.5281/zenodo.3446287.

[2] M. Abadi and contributors. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." (2015). Software available from tensorflow.org.

[3] F. Chollet et al. "Keras." (2015).

[4] D. Izzo. "Pygmo and pykep: Open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization)." In *Proceedings of the Fifth International Conference on Astrodynamics Tools and Techniques, ICATT* (2012).

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II." *IEEE Transactions on Evolutionary Computation*, **6**(2), pp. 182–197 (2002).

[6] X. Wu, T. Kozlowski, and H. Meidani. "Kriging-based inverse uncertainty quantification of nuclear fuel performance code BISON fission gas release model using time series measurement data." *Reliability Engineering & System Safety*, **169**, pp. 422–436 (2018).

[7] A. Forrester, A. Sóbester, and A. J. Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. Wiley (2008).

[8] Mathworks. "Global Optimisation Toolbox Documentation." https://uk.mathworks.com/help/gads/surrogate-optimization-algorithm.html (2019).

[9] D. Rummelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors." *Nature*, **323**, pp. 533–536 (1986).

[10] Y. Lecun and Y. Bengio. "Convolutional networks for images, speech, and time-series." In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press (1995).

[11] "Kaggle cats vs. dogs Competition." https://www.kaggle.com/c/dogs-vs-cats (2014).

[12] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:14091556* (2014).

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826 (2016).

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." *arXiv preprint arXiv:170404861* (2017).
[15] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. "The Serpent Monte Carlo code: Status, development and applications in 2013." *Annals of Nuclear Energy*, **82**, pp. 142–150 (2015).
[16] W. M. Stacey. *Nuclear Reactor Physics*. Wiley VCH Verlag GmbH (2007).
[17] N. E. Todreas. *Nuclear Systems Volume 2*. Routledge (1990).
[18] I. Dario. "Pygmo 2 Documentation." https://esa.github.io/pagmo2/ (2018). Checked: 19th September 2019.
[19] G. T. Parks. "Multiobjective pressurized water reactor reload core design by nondominated genetic algorithm search." *Nuclear Science and Engineering*, **124**(1), pp. 178–187 (1996).
[20] H. Li and Q. Zhang. "Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II." *IEEE Transactions on Evolutionary Computation*, **13**(2), pp. 284–302 (2009).
[21] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press (1996).
[22] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning." *Nature*, **521**, pp. 436–444 (2005).
[23] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." *arXiv eprint arXiv:14126980* (2014).
[24] "Supplemental information for Physor 2020." Zenodo: DOI10.5281/zenodo.3456792.
[25] A. Whyte, Z. Xing, G. Parks, and E. Shwageraus. "Design of a Deep Learning Surrogate Model for the Prediction of FHR Design Parameters." In *Mathematics and Computational Methods Applied to Nuclear Science and Engineering*. American Nuclear Society (2019).
[26] J. J. Grefenstette. "Optimization of Control Parameters for Genetic Algorithms." *IEEE Transactions on Systems, Man, and Cybernetics*, **16**(1), pp. 122–128 (1986).
[27] M. D. McKay, R. J. Beckman, and W. J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." *Technometrics*, **21**(2), pp. 239–245 (1979).
[28] I. M. Sobol. "On the Systematic Search in a Hypercube." *SIAM Journal on Numerical Analysis*, **16**(5), pp. 790–793 (1979).

APPENDIX A. CNN Topological Specifications

Table 3: Keras summary data for the CNN showing topology and modifiable parameters

Layer name (type)	Output dimensions	No. parameters
conv one (Conv2D)	(53, 53, 3)	84
pooling 1 (Average Pooling)	(26, 26, 3)	0
conv two (Conv2D)	(24, 24, 6)	168
pooling 2 (MaxPooling2D)	(12, 12, 6)	0
conv three (Conv2D)	(10, 10, 3)	165
pooling 3 (MaxPooling2D)	(5, 5, 3)	0
dense 1 (Dense)	(3000)	228000
dense 2-5 (Dense)	(250)	1528601
reshape 1 (Reshape)	(51, 51)	0