# Reducing Power Flow Simulation Cost Using Universal Function Approximators

by

**Michael Bardwell**

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

# Abstract

By integrating universal function approximators into existing simulation software, it is possible to reduce the cost of repeating simulations and thereby increase research output. In this thesis, support vector regression, random forest and artificial neural networks are deployed as universal function approximators. It is shown in an applied non-linear power flow problem that each model can achieve a maximum absolute error below 2%, and a root mean squared error below 0.2%. For selecting the number of hidden layer neurons in a single hidden layer artificial neural network, a method known as extrema equivalence is trialled. The extrema equivalence algorithm successfully identifies the approximately most sparse hidden layer size that produces near-perfect $R^2$ scores for smooth, continuous functions. Lastly, a generic file management software is proposed that can be implemented into simulation programs to save users time when re-simulating the same models with different inputs.

# Acknowledgements

I would like to thank my supervisor, Dr. Petr Musilek, for tremendous support throughout graduate school and for his dedicated work editing my conference papers and thesis, as well as for always pushing me to find opportunities. I would also like to thank Peter Atrazhev, Daniel May, Steven Zhang, Jason Wong, Carolina Quiroz Juarez and Tomas Barton for directly influencing my publications, whether through co-authoring or substantive discussions.

My parents, Sama Banaei and Ivy Naling deserve acknowledgement for putting up with my peculiarities throughout this journey. It is an honour knowing my success is their success.

Thank you to the University of Alberta for the quality working environment and engaged professors.

The support provided by Future Energy Systems under the Canada First Research Excellence Fund (CFREF) and the Natural Science and Engineering Research Council (NSERC) of Canada is gratefully acknowledged.

# Contents

# List of Figures

# List of Tables

# Abbreviations

**2D** two dimensional.

**3D** three dimensional.

**Adam** adaptive moment estimation.

**ANN** artificial neural network.

**CPU** central processing units.

**DER** distributed energy resource.

**EE** extrema equivalence.

**FEA** finite element analysis.

**FM** file manager.

**GPU** graphics processing units.

**IEEE** institute of electrical and electronics engineers.

**ILR** identity linear regression.

**LBFGS** limited-memory Broyden-Fletcher-Goldfarb-Shanno.

**LIDAR** light detection and ranging.

**MAXAE** max absolute error.

**MEANAE** mean absolute error.

**ODE** ordinary differential equation.

**PCA** principle component anaylsis.

**PF** non-linear power flow.

**PIMP** permutation importance.

**PyPSA** python for power system analysis.

**ReLU** rectified linear unit.

**RF** random forest.

**RMSE** root mean squared error.

**RW** random walk metropolis algorithm.

**SGD** stochastic gradient descent.

**SVM** support vector machine.

**SVR** support vector regression.

# Symbols

$E$ set of extrema.

$M$ cardinality of $E$.

$N$ number of samples.

$\boldsymbol{w}$ adjustable multipliers connecting neurons to each other.

$\phi$ transfer function applied to a neuron's input.

$\sigma$ basis function.

$\theta$ bias term.

# Chapter 1

# Introduction

## 1.1 Problem Statement

Modern simulation techniques are slow and repetitive. High resolution, complex simulations can take on the order of hours or days to come to completion. When these simulations are completed, it is possible the researcher or practitioner has to run another, restarting the whole process. The institutional solution to this is often to purchase faster, more expensive computers to speed up the process.

Since the outputs of a simulation are the desired results, researchers often neglect to capture the simulation model itself. The model, a mathematical mapping of input to output, is viewed as a black box, often deleted post-simulation only for the same model to be rerun later on a different input sequence. What if it was possible to mimic the relations within this box using machine learning techniques?

Machine learning and simulations are both extensively studied fields on their own, but in research areas like non-linear power flow (PF), little work has been done on meshing the two together. This thesis focuses on accurately capturing the PF simulation model, which can be applied in the electric power industry. Many institutions use simulations to make crucial decisions affecting the power vital to daily life.

It will be shown that the additional expense to the simulation is rela-

tively small. Subsequent simulation runs on the same topology are then very inexpensive, as instead of running iteration-based solvers, the user can rely on the instant output of a function map. This succeeds in alleviating slow, repetitive tasks that plague industry and academic institutions alike.

## 1.2 Capturing a Simulation Model

To briefly illustrate function mapping, imagine someone is trying to model the stress of a plate with a ball on top (Fig. 1.1).

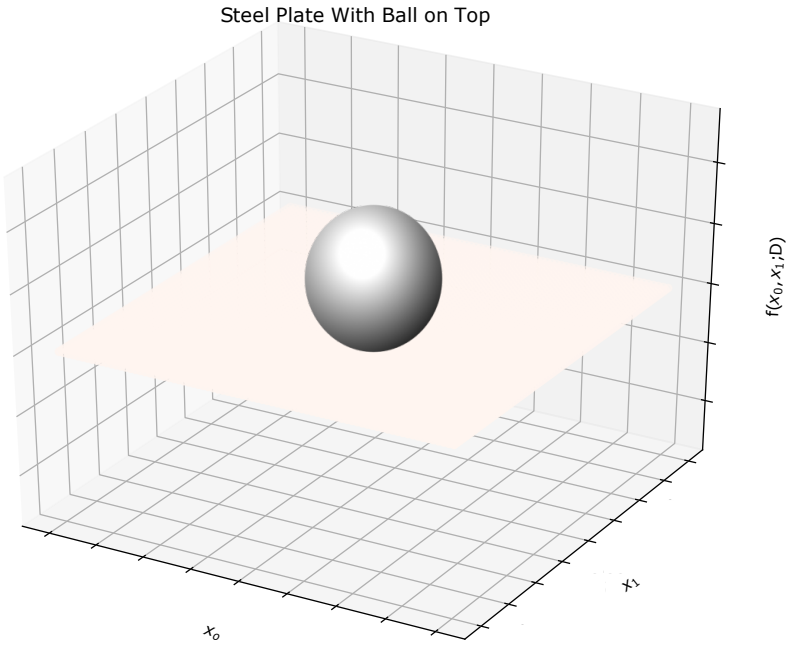
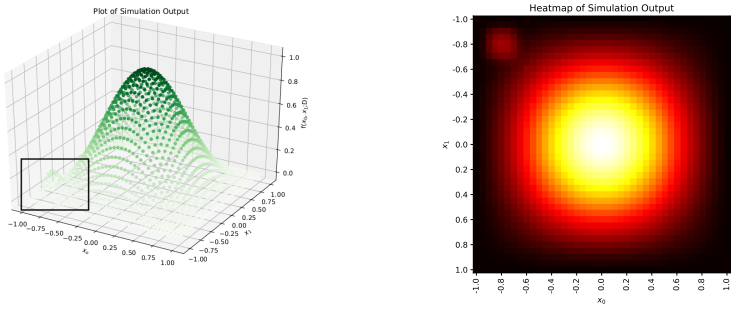



Figure 1.1: Ball on top of a square plate

The researcher places the plate and the ball in the finite element analysis (FEA) simulator; this is what we will refer to as the model. They could also input model parameters such as plate metal, the ball weight, etc... $x_o$ and $x_1$ are the independent variables, also known as inputs or features. Stress,

$S$, is the dependent parameter, also known as an output, label or target.

The researcher then runs the simulation. The FEA simulator iteratively determines the value of stress at each point $x_0$, $x_1$ and returns an output $S$, at that index. The resulting stress profile is shown in Fig. 1.2a.




(a) Plot of FEA simulation output (b) Heatmap of FEA simulation output

Figure 1.2: Finite element analysis results for stress created by ball on plate. The anomaly in the top left-hand corner of (b) is used to demonstrate how asymmetric simulation outputs can increase the complexity when selecting basis functions to approximate a simulation

The researcher notices an asymmetry in the box in Fig 1.2a and wants a closer look. They will have to create a new set of inputs, more concentrated on the area with the asymmetry and rerun the whole simulation over again. This time-costly problem is persistent in simulation-based research. Often, the solution is to buy faster and more expensive computers.

An alternative solution is to approximate the underlying mapping of the model itself (Fig. 1.2a) using a continuous function. If the approximation is good enough, it can be used in place of the simulation.

Looking at Fig. 1.2a, it is possible to deduce that

$$S = -cos(0.5\pi x_0) \cdot cos(0.5\pi x_1) \cdot e^{-(x_0^2+x_1^2)};\ x_0, x_1 \in [-1, 1] \tag{1.1}$$

perfectly fits the data collected by the researcher. The researcher can input their data into this continuous function instead of the simulation and get

the same results faster. This is, however, a very inefficient process. The researcher may spend hours guessing if the data is noisy or includes subtle asymmetries as we see in the top left corner of the FEA heatmap (Fig. 1.2b).

One automated approach to approximating functions for data sets is to assume the basis functions, $\sigma$, in a large series

$$\mathbf{S}(\mathbf{x}) = \sum^{i} a_i \cdot \sigma(b_i \mathbf{x} + \theta_i), \tag{1.2}$$

and fit the coefficients, $a_i, b_i$ and offset $\theta_i$, using a solver given an input set $\mathbf{x}$ and output set $\mathbf{S}$.

The researcher could use any combination of polynomial/periodic/-exponential basis functions, for example, to develop a useful approximation. However, there are basis functions and fitting methods that have been proven to universally approximate most bounded, smooth, multivariate functions[1],

$$\mathbb{R}^{n_{input}} \rightarrow \mathbb{R}^{n_{output}} \ : \ n_{input}, n_{output} \in \mathbb{Z}. \tag{1.3}$$

The important takeaways from this example are:

1. A model is the static mapping defined in the simulator by model parameters. It is the $\mathbb{R}^{n_{input}} \rightarrow \mathbb{R}^{n_{output}}$ correspondence between the feature and output vectors

2. If a researcher can approximate the underlying model with a continuous function, they will not need to rerun expensive simulations

## 1.3 Motivation

Why would a researcher want to approximate the output of a simulation? Is it not more convenient and accurate to use the simulation itself to produce the results? It is definitely the most accurate, but the expensive nature

[1]Multivariate describes models with multiple dependent variables, whereas multivariable describes models with multiple independent variables [1]

of simulations in both time and computational requirements can be mitigated by creating a function map. Imagine running stress simulations like in Fig. 1.3. Each simulation requires hours of calculations, so it would greatly benefit the user if they had unlimited access to an inexpensive approximation of their model.

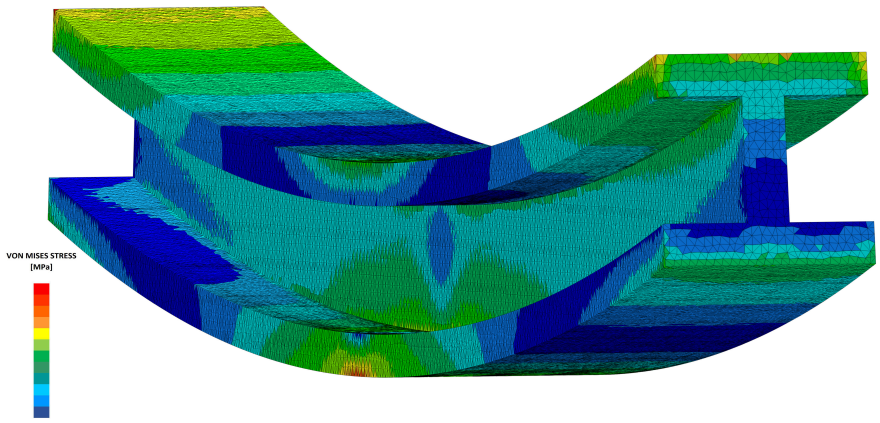



Figure 1.3: Simulation of stresses on a beam. Image use complies with Adobe Stock standard license

Now we turn our attention towards the feature application in this thesis, modern power systems.

The electric power system, or grid, is vital to modern civilization. Starting at the turn of the 19th century, companies like the Edison Illuminating Company and Westinghouse Company were providing power to thousands of local homes[2]. This has exploded into the modern grid today, annually delivering over 25551.3 TWh of electrical energy globally[3].

[2]https://power2switch.com/blog/how-electricity-grew-up-a-brief-history-of-the-electrical-grid/

[3]https://www.bp.com/content/dam/bp/en/corporate/pdf/energy-economics/statistical-review/bp-stats-review-2018-full-report.pdf

Historically, grids were designed to be vertically integrated, with market signals controlling generation facilities, which feed transmission utilities, which feed distribution companies, who supply power directly to residential, commercial and industrial consumers alike.

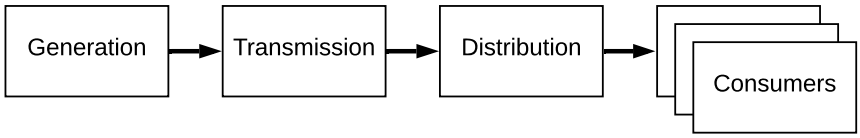



Figure 1.4: A traditional "vertical" electric grid. Consumers only receive power generated in non-local facilities

Today, with the advent of cheap and efficient distributed energy resources (DERs), grids are moving away from centralised, scheduled generation, towards distributed, non-scheduled generation[4].

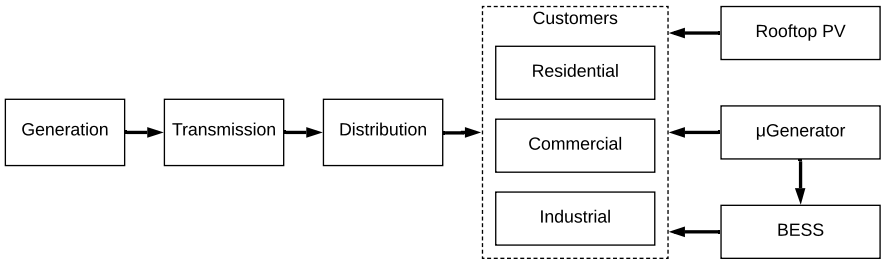



Figure 1.5: A modern "distributed" electric grid. It is different from Fig. 1.4 in that power is produced both locally and non-locally

When densely coupled with sensors and communication systems, these distributed networks become smart grids. The societal importance of smart grids is reflected in modern energy policy. There are four major objectives that drive global energy policy [2]:

[4]From Bloomberg *New Energy Outlook 2019* preview accessed August, 2019 from https://about.bnef.com/new-energy-outlook/

1. Abundant energy supply chain
2. Infrastructure to convert the supply and transmit the energy
3. Consumer cost
4. Environmental conservation

Smart grids put us in a better position to optimize for these objectives with features like:

1. Distributed energy resources (photovoltaic arrays, wind turbines, batteries)
2. Urbanised infrastructure, reducing need for long transmission lines
3. Self-producing consumers and prosumers
4. Increased grid efficiency, acute demand-side management and the adoption of electric vehicles

While smart grids come with a host of benefits, they are more complex from a power and market perspective than the traditional grid shown in Fig. 1.4. For starters, a more intensive communication network is required to link devices kilometres apart and communication latency must be factored into grid decisions. Typically, these designs are verified through co-simulations [3] - [5].

The following figures were generated to show how long simulations can get as the system becomes more complex.

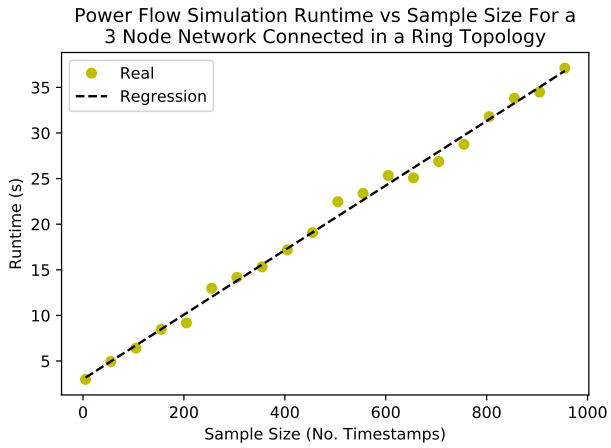



Figure 1.6: PF simulation time versus network loading sample size. Run on Intel Core i5-4210U CPU @ 1.70GHz and 16 GB of RAM

Fig. 1.6 shows that simulation runtime increases at approximately 0.034 s per timestamp. So, a simulation that wants one second worth of data at a sampling rate of 1 $\mu$s will take around $9\frac{1}{2}$ hours.

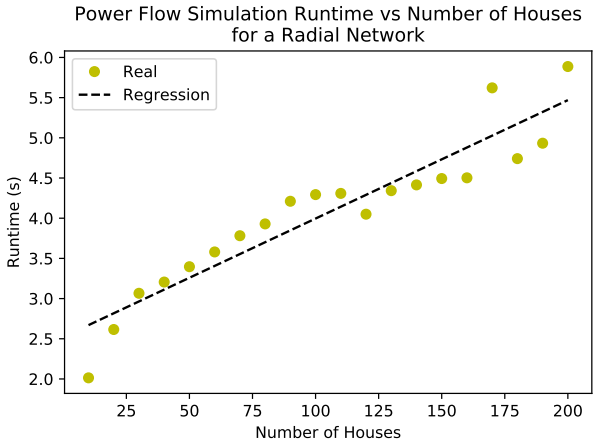



Figure 1.7: PF simulation time versus number of houses. Run on Intel Core i5-4210U CPU @ 1.70GHz and 16 GB of RAM

Fig. 1.7 shows an increase of around 0.016 s per house. A simulation of 10 million private households would take around 40 hours.

In summary, the motivation of this thesis is to show that it is possible to approximate power flow simulations. This can be used to develop a tool to reduce the need for repeated definition and execution of full power system models, replacing them with equivalent models obtained using machine learning. This tool could prove essential in institutions looking to improve the availability of simulation results (e.g. to allow for fast deduction of alternative system configurations in response to faults).

Chapter 2 surveys old and new function approximation techniques and other simulation improvement approaches.

Chapter 3 outlines the essential background knowledge in machine learning required for the project. The problem is broken down into manageable pieces and the main application, PF analysis, is discussed.

Chapter 4 presents the results for key metrics defining each checkpoint outlined in Chapter 3. The effectiveness of each approach is discussed.

Chapter 5 proposes a general method to implementing automated approximations into simulation software.

In Chapter 6, the major claims made in this thesis are re-affirmed and future work is outlined.

This thesis will show that it is possible to approximate power flow simulations, such that the approximation captures the trend of the input/output correspondence. It will be demonstrated that an analytical method to select neural network hidden layer density produces excellent results for multi-dimensional, smooth, sinewaves. Finally, the computational cost of inserting function approximation and metadata storage into pre-existing power flow simulation software is shown.

# Chapter 2

# Related Work

## 2.1 Function Approximation

Approximation is differentiated into two categories: regression and interpolation. Interpolation, by definition, fills the space between samples according to predefined rules whereas regression minimises a cost function. The latter permits generalisation of a data set, which is useful for applications with limited or noisy training data.

Whether a model should be approximated using regression or interpolation depends on how precisely that data used to train or fit the model represents the whole data set. Typically, the root-finding iterative solvers underlying simulations try to minimize error below a specified threshold, within a predetermined number of iterations. Despite ensuring the best accuracy, coupling a long simulation with expensive interpolation techniques can become too costly for many users. For time-sensitive use cases, tolerance can be set higher at the cost of precision. For example, in the case of ordinary differential equation (ODE) solvers such as Runge-Kutta techniques [6] used in MATLAB Simulink[1], low tolerance simulations can take days. For faster results, users can increase the tolerance. In this scenario, interpolation would then overfit the less precise data, whereas regression can be used to develop a generalised model.

[1]https://www.mathworks.com/help/simulink/ug/types-of-solvers.html

This thesis is an extension of the work presented in [7]. A core finding in the work is that identity linear regression (ILR) outperforms an artificial neural network (ANN) when approximating the voltage profiles of a 3-node network computed using the python for power system analysis (PyPSA) PF solver. In future work, the paper suggests that extrema be considered for analytically selecting the size of the hidden layer. The work also suggests testing deep rectified linear unit (ReLU) networks. Other popular regression methods will also be considered.

Hornik et al. propose that for every Borel measurable, $\mathbb{R}^n \rightarrow \mathbb{R}^m$ function $f$, there exists a multilayer feedforward network capable of approximating $f$ to any degree of accuracy [8]. Park and Sandberg prove Radial Basis Functions are also universal approximators [9]. This is applied, for example, by Zainuddin and Pauline on periodic, exponential and piecewise continuous functions [10]. One issue not discussed in the seminal group of papers was the functional approach to selecting hidden layer density for a single hidden layer ANN. This is tackled by Zhang et al. in [11], whose theory is explained in chapter 3 and experimentally tested in chapter 4.

ANNs are commonly used for classification in academia like [12], where ANNs were trained to estimate the likelihood of deadly debris flow given a satellite image of the terrain. May and El-Shahat used ANN-based regression to model battery degradation [13]. Estimating surface roughness given machining parameters is also a popular application [14] - [16].

Support vector machines (SVMs) have become popular because they work well on small data sets. Support vector regression (SVR) is used to model a three-dimension microwave packing structure [17] using the radial basis function kernel,

$$K(x, x_i) = exp(-\gamma \cdot \|x - x_i\|^2). \tag{2.1}$$

The same group trained an SVR to produce accurate asphalt concrete permittivity [18]. The simulation software in both papers was Ansoft HFSS. Similarly in [19], SVR is deployed to model hysteresis curves of an Alu-

minum Nickel Cobalt Alloy. SVR was also used to generate fine-resolution maps from coarse fractional images by fusing data sets such as: panchromatic images, digital elevation models, light detection and ranging (LIDAR) data multiple subpixel shifted images and prior information of buildings [20].

Random forest (RF), a extension of the decision tree concept, has also been used for regression [21]. RF regression was used in [22] to build a ultra-responsive space vector pulse width modulation controller. RF bias is reduced using a boosting technique in [23], leading to improvements over the original RF method of up to 82%.

Each regression method requires a solver to minimise its cost function. For ANNs, one of the more popular variants is stochastic gradient descent (SGD) [24]; an optimization technique applied to problems such as speech recognition [25]. Kingma and Ba develop an adaptation of AdaGrad and RMSProp [26] called adaptive moment estimation (Adam). Adam works with dynamic objectives, sparse gradients and naturally adjusts its step size using first and second moments of the gradient[2].

Much of the recent applications of regression have been SVR and RF based; likely because they have less parameters to tune and do not get stuck in local minima. ANNs are studied in this thesis because their potential capacity makes them a useful tool for function approximation. There is also a substantial amount of ANN usage in classification problems, leading to large training algorithm improvements that may be harnessed in the future.

## 2.2 Optimizing Simulations

There are many ways to enhance simulations. Some approaches aim to reduce the amount of data handled by the simulator. Some parallelise the work to analyse multiple features in one shot. Others optimise hardware management to efficiently schedule task execution.

Reducing the amount of data handled by a simulation is largely divided

[2]https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

into two streams: modifying model component behaviour and changing component form. To identify areas for simplification, however, requires embedded tracing methods. A manufacturing example in [28] orders machines by utilisation and converts the least used to constants. This method is slow, however, because a human has to identify the heuristics for "utilisation".

Watson and Arrillaga discuss mixed time-frame simulation [29]. In order to capture the long-time, constant nature of a generator (typically solved using 200 $ms$ steps) in an electromagnetic simulation (typically solved using 50 $\mu s$ steps) efficiently, the NETOMAC simulator integrates instantaneous and stability modes at runtime. They also describe the very simple, sparse-matrix technique, in which only non-zero elements are stored and accessed, therefore reducing cost.

Larger projects, such as in aerospace engineering, require simulations of virtual, multi-disciplinary prototypes. These heterogeneous models are an amalgamation of simulations from independent domains, with each domain necessitating massive resources for timely completion. The authors in [30] employ a distributed interactive environment on the cloud to concurrently run independent models, reducing the simulation time of 100 cases from 6958 seconds to 172 seconds. This work could be extended by considering the use of CloudSim to estimate scheduling and migration performance for heterogeneous datacenters [31].

Some algorithms take a hardware approach to simulation cost reduction. Integrating critical channel traversing on the TasKit kernel demonstrates two-to-three time speed improvements when compared to a splay tree central-event-list based sequential kernel [32]. This is achieved through automated load-balancing, regimenting cache-behaviour and multi-level scheduling.

There are many ways to optimise simulation software to reduce cost. Often researchers use GitHub issues to request new features, like permitting asymmetric impedances or adding non-linear direct-current power

flow equations to PyPSA[3]. Other methods require a data-reduction or hardware approach as described above. The remaining chapters prove the viability of a regression-based method, which when mixed with other ideas can lead to a significant reduction in total simulation time.

[3]https://github.com/PyPSA/PyPSA/issues

# Chapter 3

# Background

## 3.1 Function Approximation

### 3.1.1 Artificial Neural Networks

A fully-connected, feedforward ANN, shown in Fig. 3.1, is a category of model under the machine learning umbrella. It is defined as a specific combination of input, hidden and output layers. The foundation of ANNs, perceptrons, were initially proposed by Rosenblatt in the 1958 [33]. ANNs come in a variety of widths and depths.

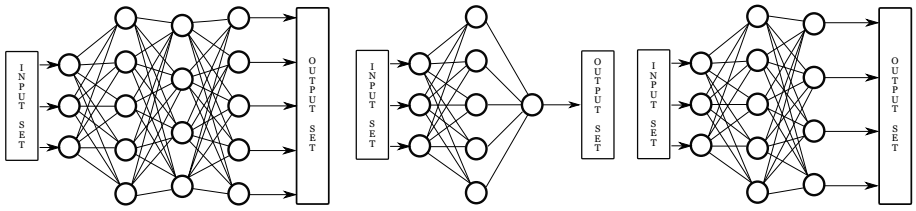



Figure 3.1: Feedforward, fully-connected, ANN topology examples. Each circle is a computational unit or neuron. (a) has two hidden layers and five outputs. (b) has one hidden layer and one output (c) has one hidden layer and four outputs

The notation $a_c^b$ denotes $b$ and $c$ as orthogonal iterators, with $b$ iterating

over columns and $c$ over rows, for example. Indexing starts at one unless otherwise noted. Any approximation with be denoted with a hat (e.g. $\hat{y}$).

For a fully connected network, in each layer, $l^k$, lies a set of neurons, $n$, otherwise known as computational units, or nodes. To calculate the output for each node, $i$, one must know the weight vector $\boldsymbol{w}_i^k$, bias $b_i^k$ and input $\boldsymbol{x}_i^k$ to the node[1]. To calculate the output of neuron $j$ in $l^k$ first find $h_j^k$

$$h_j^k = \boldsymbol{w}_j^k \cdot \boldsymbol{x}_j^k + b_j^k = \sum_{i=1}^{n^{k-1}} w_{i,j}^k \cdot x_{i,j}^k + b_j^k. \tag{3.1}$$

An activation function, $\phi$, is then applied as follows to produce an output

$$o_j^k = \phi(h_j^k). \tag{3.2}$$

Note the output vector, $\boldsymbol{o}^{k-1}$ is used as the input vector $\boldsymbol{x}^k$ between $\{l^2, ..., l^m\}$. To calculate the ANN output, $\hat{\boldsymbol{y}}$, the outputs for each layer are computed up to the last layer, $l_m$. $\hat{\boldsymbol{y}}$ is then typically computed using a linear activation function $\phi(h) = h$,

$$\hat{\boldsymbol{y}}_i^m = \{h_i^m, h_{i+1}^m, ..., h_n^m\}. \tag{3.3}$$

Calculating the output $\hat{\boldsymbol{y}}$ is known as a forward pass. For a set of ordered pairs $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, each sample, $N$, can be calculated in parallel and summed at the end. This is why training is typically done using hardware tailored for parallelisation (ex: graphics processing units (GPU), multicore central processing units (CPU), computer clusters) [34] [35]. The same is true for the backward pass.

For hidden layers, the activation function, $\phi$, is typically a monotonically increasing, bounded differentiable function. In this thesis, a sigmoid function [36], sometimes referred to as a squashing function [37] or logistic function[2] is defined as

[1] https://brilliant.org/wiki/feedforward-neural-networks/#formal-definition
[2] https://en.wikipedia.org/wiki/Logistic_function

$$\phi = \frac{1}{1 + e^{-x}}. \tag{3.4}$$

The sigmoid function belongs to a set of sigmoidal functions [37], [38] that are very common in ANNs. A function is sigmoidal if for $\phi : \mathbb{R} \to \mathbb{R}$

$$\lim_{x \to -\infty} \phi(x) = 0 \ \textit{and} \ \lim_{x \to \infty} \phi(x) = 1; \tag{3.5}$$

it is bounded and monotonically increasing.

A univariate ANN, where $\hat{\boldsymbol{y}} \in R$, is seen in Fig. 3.1 (b). These are the most simple functions to prove the methodology with and will be approached first. After, multivariate functions, like in Fig. 3.1 (c); $\hat{\boldsymbol{y}} \in \mathbb{R}^4$, will be approximated.

**Training**

Training is the process of adjusting the network weights and biases such that the model can better approximate $D$. Training is typically performed using backpropagation which requires an error function, $\mathcal{E}$. This thesis uses the popular mean squared error function,

$$\mathcal{E}(X) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2. \tag{3.6}$$

To determine which direction the weights need to be adjusted, the gradient of $\mathcal{E}$ with respect to $w$ is required. If we denote $a_i^k = \sum_{j=0}^{n_{k-1}} w_{i,j}^k \cdot o_j^{k-1}$, where $w_{i,0}^k = b_i^k$ and $o_0^{k-1} = 1$ then

$$\frac{\partial \mathcal{E}}{\partial w_{i,j}^k} = \frac{\partial \mathcal{E}}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{i,j}^k} = \delta_j^k \frac{\partial a_j^k}{\partial w_{i,j}^k}. \tag{3.7}$$

Where $\delta_j^k$ is often referred to as the error signal. Using this formula, it is possible to derive the gradient for any layer in the network. For example, the equation for the final layer, $m$, is

$$\frac{\partial \mathcal{E}}{\partial w_{i,1}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y)\phi'(a_1^m) o_i^{m-1}, \tag{3.8}$$

and for the hidden layers, $l_1 - l_{m-1}$ is

$$\frac{\partial \mathcal{E}}{\partial w_{i,1}^m} = \delta_j^k o_i^{k-1} = \phi'(a_j^k) o_i^{k-1} \sum_{z=1}^{n^{k+1}} w_{j,z}^{k+1} \delta_z^{k+1}. \tag{3.9}$$

Because of the sum rule in differentiation, it is also possible to parallelise backpropagation. This makes deep ANNs more viable, since the training cost can be divided across multiple computational units, for example.

Before training starts, ANN weights must be properly initialised. It is often done according to insight from He et al. [39] and Glorot and Bengio [40]. The magnitude of the weights should be inversely proportional to the number of input connections (or fan-in) to reduce overshooting. The input weights for each node are initialized randomly according to a zero-mean Gaussian distribution with a standard deviation of $\frac{2}{\sqrt{n^{k-1}}}$. Each node is given a bias for affine transformation, initialised to zero.

ANNs are notoriously hard to parameterise, specifically their width and depth. A novel technique to select the number of neurons for the hidden layer in a single hidden layer ANN is introduced below.

**Extrema Equivalence**

Zhang et al. [11] argue that for any continuous function, $f(x)$, defined on a compact set $C \in \mathbb{R}^n$, a single hidden-layer ANN with sigmoidal activation, can create an extrema equivalent function, $g(x)$. Extrema equivalence (EE) is denoted as $f(x) \bowtie g(x)$ and means both $f(x)$ and $g(x)$ contain the same set of extrema, $E = \{(\mathbf{x_1}, f(\mathbf{x_1}), \ldots, (\mathbf{x_M}, f(\mathbf{x_M}))\}$. For multivariable functions the residual of $|f(x) - g(x)|$ is bound such that

$$|f(\boldsymbol{x}) - (\sum_{i=1}^{M} c_i \tilde{\sigma}(\boldsymbol{w}_i \boldsymbol{x} + \theta_i) + c_0| < \epsilon \quad \forall x \in E_L \cup [a, b]. \tag{3.10}$$

$M$ is defined as the number of extrema in $f, g$. $\epsilon$ is a constant. $c_i$ is the output weighting for each neuron and $c_0$ is a bias term. $w_i$ represents the fan-in weight vector. $\theta_i$ is the bias term for each neuron.

Note that for univariable functions, Zhang et al. prove $M + 1$ hidden layer neurons are required to create an extrema equivalent function. It is also important to emphasise that the work only claims to approximate the points in $E$ within the given tolerance. Thus, the algorithm is a starting point for evaluating which hidden layer size will produce the best approximation, with minimal hidden layer neurons, of the total data set.

To calculate M we must first define *extremum*. A point, $\boldsymbol{x}_i$ is considered an extremum if the following inequalities are satisfied:

$$f(\boldsymbol{x}_i) > max\{f(\boldsymbol{x}_{k_1}), f(\boldsymbol{x}_{k_2}), ..., f(\boldsymbol{x}_{k_i})\} + \eta \tag{3.11}$$

$$f(\boldsymbol{x}_i) < min\{f(\boldsymbol{x}_{k_1}), f(\boldsymbol{x}_{k_2}), ..., f(\boldsymbol{x}s_{k_i})\} - \eta \tag{3.12}$$

where $\{\boldsymbol{x}_{k_1}, \boldsymbol{x}_{k_2}, ..., \boldsymbol{x}_{k_i}\}$ forms a polytope surrounding $\boldsymbol{x}_i$. This algorithm was implemented in Python and the results are demonstrated in chapter 4.

To demonstrate the capacities of different hidden layer sizes, the results of single hidden layer ANNs with $n_{hidden\ layers} = 1, 2, 3, 4$ were plotted in Fig. 3.2. The ANN in the top left figure was trained on a single neuron, meaning that the network was a simple sigmoid function. The curve of the sigmoid activation function is obvious; the negative bias has vertically translated the function such that the upper bound of the sigmoid function, $y_{x\to\inf} \approx 0.5$. At $n = 2$, we can see two sigmoidal shapes converging. At $n = 3, 4$, the sigmoidal shape disappears and the network begins to look like the original model.

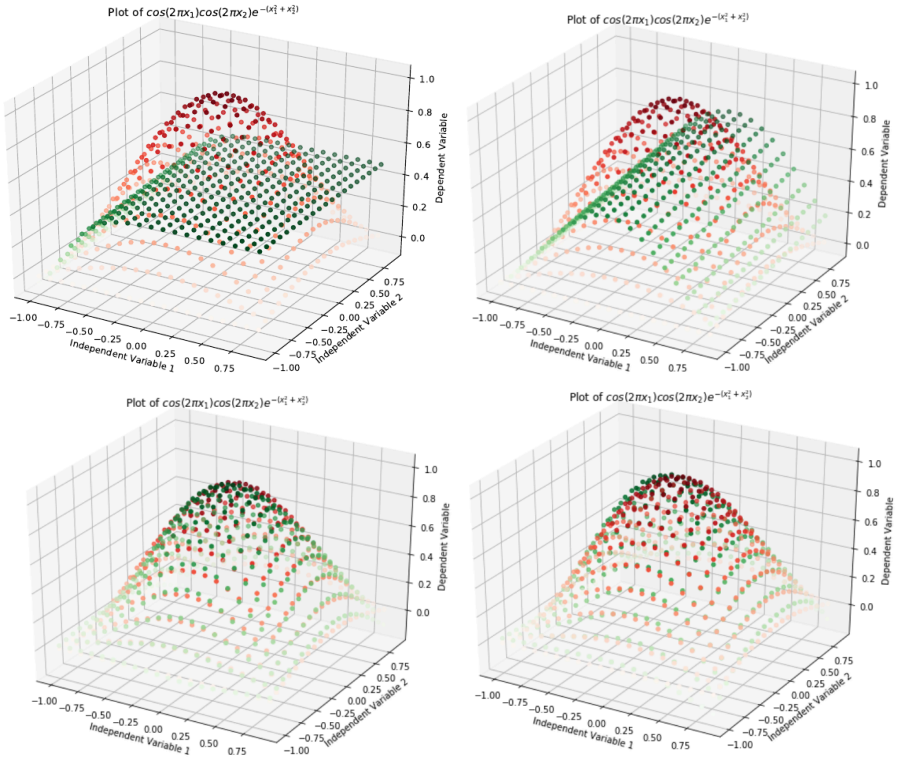



Figure 3.2: Overlay of actual and approximated values of limited capacity ANNs. Keeping everything else the same, the number of hidden layer neurons was 1: top left, 2: top right, 3: bottom left, 4: bottom right. Each network was fed 400 samples

### 3.1.2 Support Vector Machines

SVR, an application of SVMs was introduced in 1995 by Cortes and Vapnik [44]. SVR training operates on the structural risk minimum. The idea is to map vectors into high-dimensional space $\mathbf{Z}$ using non-linear functions and then find the optimal separating hyperplane with maximum margins. For example, if

$$\boldsymbol{w}_o \cdot \boldsymbol{x}_i + b = 0 \tag{3.13}$$

is the optimal hyperplane between two classes, the hyperplane ($\boldsymbol{w}_o$, $b_0$)

maximises the distance $\rho$ between the support vectors in each class,

$$\rho(\boldsymbol{w_0}, b_0) = \frac{2}{\sqrt{\boldsymbol{w_0} \cdot \boldsymbol{w_0}}}. \tag{3.14}$$

More generically[3], for a linear function $f = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$,

$$\begin{aligned} &minimise \quad \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \\ &subject\ to\ \forall\ i: \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b & \leq \epsilon + \xi_i \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases} \\ &|\xi|_\epsilon = \begin{cases} 0 & if |\xi| < \epsilon \\ |\xi| - \epsilon & otherwise, \end{cases} \end{aligned} \tag{3.15}$$

where $f(x)$ has at most $\epsilon$ error from the sample $y$, $C$ is the penalty parameter that offsets the number of deviations larger than $\epsilon$ with keeping $w$ small (aka $f$ flat). $\xi_i, \xi_i^*$ are slack variables, analogous to "soft margins". The cost function in (3.15) is a convex optimisation problem, which, unlike many ANN scenarios allows for solvers to always find the global solution [45].

### 3.1.3 Random Forest

Random Forest (RF) is an ensemble method introduced by Breiman in 2001 [21]. The algorithm is described as "a collection of tree-structured classifiers $h(\mathbf{x}, \Theta_k) : k = 1, ...$ where $\Theta_k$ are independent, identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$".

Assuming the training set is randomly drawn from the data set $D = \{\mathbf{X}, Y\}$, the MSE is given as

[3] https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html

$$E_{\mathbf{X},Y}(Y - h(\boldsymbol{x})^2), \tag{3.16}$$

where $h(\boldsymbol{x})$ is any numerical predictor. To form the predictor, the algorithm takes an average over $k$ of the trees $h(\mathbf{x}, \Theta_k)$.

It is noted by the RF creator that for regression, RF correlation increases slowly with number of features. Therefore, a large number of features are required to reduce test-set error. Li et al. [46] propose a multinomial RF framework to fix the gap in Breiman RF which sees results not optimally converging as sample size increases.

### 3.1.4 Model Generalisation Error

After a regressor is trained, its accuracy and precision are determined by various heuristics. If the model fails to capture the overall trend of the data, it is deemed underfitted. One common error that leads to underfitting is poor parameter selection; it is hard, for example, to capture a line with a parabolic model.

For ANNs, one approach to ensuring that the model will not underfit is to increase the *capacity*. This is often achieved by making the network deeper (more layers) and wider (more neurons per layer). The issue with capacity is that biasing decreases and variance grows as model capacity increases [48], which leads to overfitting. The failures of adding capacity to solve underfitting are explored in [49]. The authors in [50] find that for larger networks both bias and variance decrease with network width. To solve overfitting, a regularisation term is typically added.

SVRs also rely on adding regularisation terms to their cost function to prevent overfitting. For RFs, overfitting is mitigated by averaging multiple decision tree models. It is more likely that SVM and RF suffer from model selection overfitting caused by errors in the model selection tools like $k$-fold cross validation [51].

Model generalisation boils down to two metrics

1. Bias: the accuracy of the model
2. Variance: the precision of the model.

Getting a well generalised model is heavily dependent on the way data is used during the model construction process. Sample selection and sectioning is the first step in any training process.

### Selecting Number of Samples

Before training a model, users must have a basic understanding of how many samples, $N$, are available to them, and how many they need. Raudys and Jain discussion on pattern recognition systems for two class problems concludes that $N$ should scale with the number of features and the complexity of the decision rule [54]. For simulation approximation in this thesis, $N$ is selected via an experimental approach, where $N$ is increased by order of magnitude until the score saturates or the memory requirements are too large for the computer. In MATLAB's function approximation class[4], the discretised points of which there are $N$, are referred to as breakpoints. The class provides the user with the option to explicitly specify breakpoint locations, or to use even spacing.

### Training, Validation and Test Sets

After selecting $N$, the next step to training a model is to split the data into training, validation and test sets. First, a test set should be split off to avoid data snooping bias. Guyon recommends the size of the test set be inversely proportional to the probability of error, $p$ [55]. For example, Guyon looked at the error rate for digit classifiers and found the best models had a rate under 1%. After making a few assumptions, the authors suggested the following model

[4]https://www.mathworks.com/help/fixedpoint/examples/fixed-point-function-approximation.html

$$n \simeq \frac{100}{p}. \tag{3.17}$$

Where $n$ is the number of samples. For the digit classifier, the test set should have 10 000 samples.

The proper percent of the remaining data that should be allocated to training versus validation was studied in a sister paper by Guyon [56]. The author recommends scaling the validation-to-training set ratio according to

$$\sqrt{\frac{ln\ N}{h_{max}}}, \tag{3.18}$$

where N is the size of the family of recognisers (e.g. how many ANNs are being tested) and $h_{max}$ is the largest complexity within those families (e.g. number of parameters in an ANN). Often, an 80/20-80/20 rule is recommended, where 20% of the full data set is test data and 20%/80% of the remainder is allocated to validation/training data respectively.

#### k-Fold Cross Validation

It is possible for two models to both achieve high scores when trained on one data set, so which is better? One method to determine the best model is $k$-fold cross validation [57]. A shuffled set is divided into $k$ folds (or groups) and each combination of $k$-1 folds is used as the training set.

| | Fold 1 | Fold 2 | Fold 3 |
|---|---|---|---|
| Round 1 | Validation Set | Training Set 1 | Training Set 2 |
| Round 2 | Training Set 1 | Validation Set | Training Set 2 |
| Round 3 | Training Set 1 | Training Set 2 | Validation Set |

Table 3.1: $k$-fold cross validation

Each set is scored against the outlier fold. Assuming the data set is large enough that each fold would be a representative sample of the data set, the model with the most consistent scores (lowest standard deviation) is likely the best. The value of $k$ is usually chosen to be 5 or 10, which has

been found through experimentation to yield reasonable variance/bias[5]. A counter argument to this is presented in [51], where it is shown that variance between k-fold estimations is commonly underestimated in model selection criterion leading to poor selection.

## 3.2 Data Analysis and Manipulation

### 3.2.1 Correlation

Before training a model, it is important to analyse the available data sets for relationships. First, the relation between features should be calculated using bivariate correlation techniques. Two popular techniques include Pearson's $r$,

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{3.19}$$

and Kendall's $\tau$,

$$\tau = \frac{n_c - n_d}{\binom{N}{2}}, \tag{3.20}$$

where $n_c$ is the number of concordant pairs and $n_d$, the number of discordant pairs. To calculate $n_c$, go element-by-element two vectors at a time; if $sign(A_i - A_{i-1}) = sign(B_i - B_{i-1})$ add one to $n_c$, if not add one to $n_d$. For example, if $A = \{1, 2, 3\}$, $B = \{1, 3, 2\}$, $n_c = 1$, $n_d = 1$ so $\tau = 0$.

$r$ provides a measure for how linearly related two variables are, assuming that both are normally distributed and homoscedastic[6] (e.g. each sample is the same distance from a fitted regression line). $\tau$ is a measure of the ordinal association between variables. In other words, $\tau$ represents the probability that the data is in the same order versus not in the same order [58].

The next step in analysis is to calculate the relationship between fea-

[5]https://machinelearningmastery.com/k-fold-cross-validation/

[6]https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/

tures and labels (or targets). The aforementioned correlation techniques can be used. Another method to determine the importance of features to a model is known as permutation importance (PIMP) [59]. To implement PIMP, train a model normally and calculate a score. Then, shuffle the data randomly for one feature at a time and re-calculate the score. The features, that when shuffled, degrade the score the most are evidently more important to the prediction power of the model.

Out of all available (reasonable) features, often some are not included in training because they lack correlation with the target set. When deciding to cut features, it is important to keep in mind that statistical models will substitute the impact of missing variables with those included. This is known as omitted variable bias [60].

### 3.2.2 Dimensionality Reduction

Once it is determined that certain features are unwanted, whether because they are too related to other features or unrelated to the labels, they can either be removed completely from the data or merged using principle component anaylsis (PCA) [61]. With less features, models should train faster, but in some cases removing features reduces model performance.

### 3.2.3 Scoring

Scoring has been used thus far as a generic term synonymous with model performance. Scores are how users evaluate the precision and accuracy of a machine learning model. Scoring can be split into runtime and post-run categories.

Measurements must be made during training to gain insight into the process. For example, *sklearn*'s MLP Regressor class uses squared loss to make runtime decisions. Squared loss functions are especially sensitive to increases in the variance of the frequency distribution of outliers, which is useful for approximating simulation (solver-based) data which would only contain outliers in rare non-convergent cases.

Post-run scores are often referred to as skill scores, with the coefficient of determination, $R^2$, being the most common. $R^2$ depicts how much the model explains the variability of the original data.

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}(f_i - \hat{f}_i)^2}{\sum_{i=0}^{N-1}(f_i - \bar{f})^2}, \tag{3.21}$$

where $\bar{f}$ is the mean of $f$ (the function to be approximated) and $\hat{f}$ is the approximation. A score of 1 means the model explains the variability well. In code, this can be connected to a Python *Assert* statement, for example, to act as a litmus test. Some literature has recommended the adjusted skill score which offsets the phenomenon where skill score naturally tends to increase with the number of features [62].

For error analysis, the root mean squared error (RMSE) is used because it is in the same units as the outputs but still penalises outliers like MSE,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{f}_i - f_i)^2}{N}}. \tag{3.22}$$

Other error measures used in this work include mean absolute error (MEANAE)

$$MEANAE = \frac{\sum_{i=1}^{N}(\hat{f}_i - f_i)}{N}, \tag{3.23}$$

and max absolute error (MAXAE)

$$MAXAE = max\{(\hat{f}_1 - f_1), \ldots, (\hat{f}_i - f_i)\} \quad \forall i \in N. \tag{3.24}$$

Scores become important when mapping correspondences over three dimensions. It becomes hard to visualise the results past three dimensions and therefore scores are relied upon to tell the user if the model is good or not. In chapter 4, heuristics for each score are determined by reasonable assertions based on the desired application; arguments are presented for quantitative values which researchers in the field would consider usable.

## 3.3 Power System Load Flow Simulations

PF simulations are used for a variety of reasons. High sampling rate simulations are used to capture electromagnetic events like line shorting for $n$-1 tests or circuit breaker closures to determine power-on transients. Slower events like generator spin ups and power changes in large-inertia systems like wind-turbines are captured using less frequently sampled simulations. A commonly calculated property is bus voltage, which since the voltage at each bus is relative to other buses, is an iteratively derived solution.

When the simulator starts, it initialises the loss function using the model, and ports load information from a data set into the calculator. The simulator then iteratively searches for the optimal solution. For power system load flow (PSLF) programs, these solvers use techniques like Newton-Raphson or Gauss-Seidel as seen in Fig. 3.3 to solve the non-linear equation,

$$V_i = \frac{1}{Y_{ii}} \cdot \left( \frac{S_i^*}{V_i^*} - \sum_{k=1,k\neq i}^{n} Y_{ik} \cdot V_k \right). \tag{3.25}$$

Where $Y \in \mathbb{C}^n$ is the networks bus admittance matrix, $S \in \mathbb{C}^n$ is the apparent power matrix and $V \in \mathbb{R}^n$ is the bus voltage matrix. Individual buses are represented by iterators $i, k$.

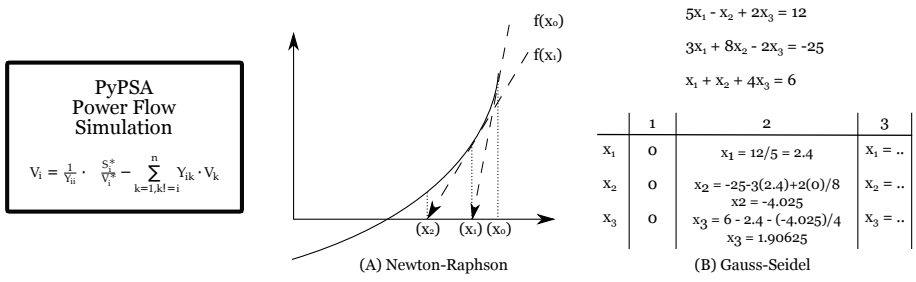



Figure 3.3: An example of (A) Gauss-Seidel and (B) Newton-Raphson iterative root-finding methods

The iterative process can be avoided for repeated models by, for example, capturing the mapping between the load and voltage information in

Eq. (3.25). Once captured, these approximations would have to be stored using a file manager for re-importing in the future.

In this chapter, function approximation models, the extrema equivalence algorithm for ANNs, data analysis and the math behind power flow simulations were introduced. In the next chapter, EE is tested for its approximation capability and runtime. PF simulations are then approximated. Lastly, a framework for automating function mapping in simulation software is presented.

# Chapter 4

# Experimental Results and Analysis

## 4.1 Extrema Equivalence

### 4.1.1 Problem and Objective

Currently, ANN training primarily relies on search techniques for selecting optimal network parameters [42]. Search techniques are slow, because many models have to be trained, an often expensive process.

Zhang et al's extrema equivalence (EE) algorithm provides a reasonable analytical starting point for optimising the hidden layer size for a single-hidden layer feedforward ANN (simply referred to as ANN in the rest of §4.1). The cost and accuracy of the technique is evaluated for potential use in future experiments. Specifically, the following should occur in less time than any reasonable hidden layer size search:

1. the EE algorithm must find all members of the extrema set $E$. The cardinality of this set is $M$

2. using the $M/M+1$ rule as the starting point for a hidden layer size search, the "guided search" must find the optimal[1] number of neurons

[1] The metric for optimal is application dependent (e.g. meeting an RMSE or $R^2$ threshold)

### 4.1.2 Introduction

Uniform distributions can be visualised in 2D matrix form.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1N} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & x_{N3} & \dots & x_{NN} \end{bmatrix}$$

Each point that is surrounded by four neighbours in a square is considered a potential extrema candidate per Eq. 3.11/Eq. 3.12. The EE algorithm must compare $N$ points with $2^n$ neighbours. The time-cost of the extrema algorithm increases at the rate

$$O(N \cdot 2^n). \tag{4.1}$$

$N$ can be replaced with $(N_i)^n$ in a uniform distribution, where $N_i$ represents the number of samples in any dimension. The total complexity for the test set is then shown as

$$O((N_i)^n \cdot 2^n). \tag{4.2}$$

The tests are run in the following sequence,

1. Verify that the extrema finder produces the right set $E$
2. Approximate a series of functions using $M/M+1$ as a guide
3. Gauge how EE runtime scales with $n$, $N$

### 4.1.3 Results

The following univariate decaying sinewave function was used to test the algorithm,

$$f(x_i, x_{i+1}, ..., x_n) = \prod_{i=0}^{n} sin(a\pi x_i)e^{-x_i}, \tag{4.3}$$

where $a$ is a user-defined constant. The resulting number of extrema, $M$, for decaying sinewaves of various dimensionality is tabulated.

| $n$ | Create Data Set (s) | $M$ | Find Extrema (s) |
|---|---|---|---|
| 1 | 0.0003 | 3 | 0.0002 |
| 2 | 0.0166 | 9 | 0.0089 |
| 3 | 0.9642 | 27 | 0.4872 |
| 4 | 52.3575 | 81 | 25.2188 |

Table 4.1: Number of detected extrema, $M$, for $n$-dimensional decaying sinewaves defined in Eq. 4.3; $a = 1$. "Create Data Set" is the amount of time required to create the input and output data sets. "Find Extrema" is the algorithm run time

These results are visualised for $n = 1,\ 2$ in Fig. 4.1 and 4.2, respectively. As expected, both figures show extrema at the peaks of the data set, and the number of extrema scales exponentially at the rate

$$M_n = (M_{n=1})^n. \tag{4.4}$$

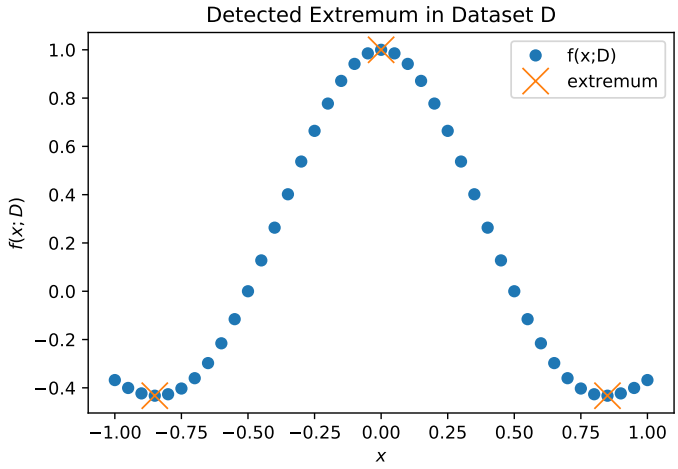
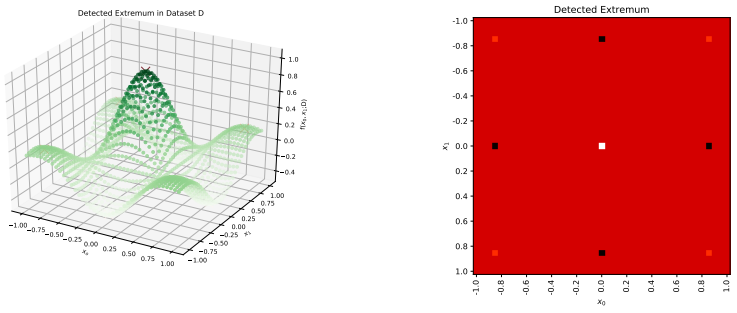


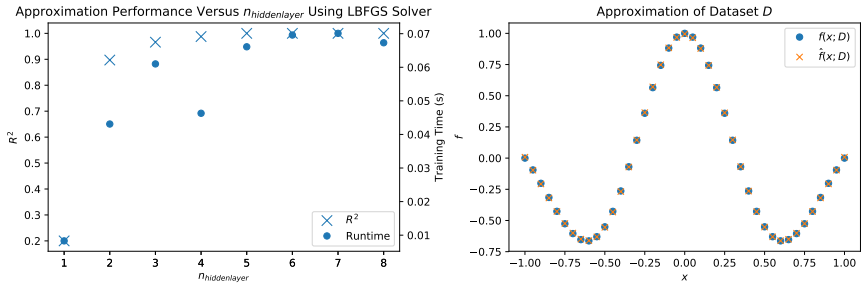

Figure 4.1: Detected extrema in two dimensional (2D) data set using EE algorithm




(a) (b)

Figure 4.2: Detected extrema in three dimensional (3D) data set using EE algorithm. (b) is a heatmap of (a)

Next, data sets drawn from various functions were approximated. $n_{l_1}$, $n_{l_2}$ and $n_{l_3}$ refer to the size of the input, hidden and output layers respectively. Function mapping techniques were applied to $n_{l_1} = 1,\ 2$ data sets. This makes it easy to visualise the results.




(a) (b)

Figure 4.3: (a) Approximation performance in the form of $R^2$ and training time against $n_{l_1}$. (b) Overlay of the real set $D$ and the approximated values. In $D$, there are three extrema ($M = 3$). The approximated function is Eq. 4.3 with $a = 1.5$ and $N = 40$

First, various hidden layer sizes are trialed to approximate the function in Fig. 4.3b. The approximation is expected to be almost perfect around $n_{l_2} = M$. Indeed, the score saturates around $n_{l_2} = 3$, which has a value of

0.98 as seen in Fig. 4.3a.

Next, a logarithmic function was tested.

$$f(x_i, x_{i+1}, ..., x_n) = \prod_{i=0}^{n} log_{10}(|a \cdot x_i|). \tag{4.5}$$

A single extrema was detected near the origin.

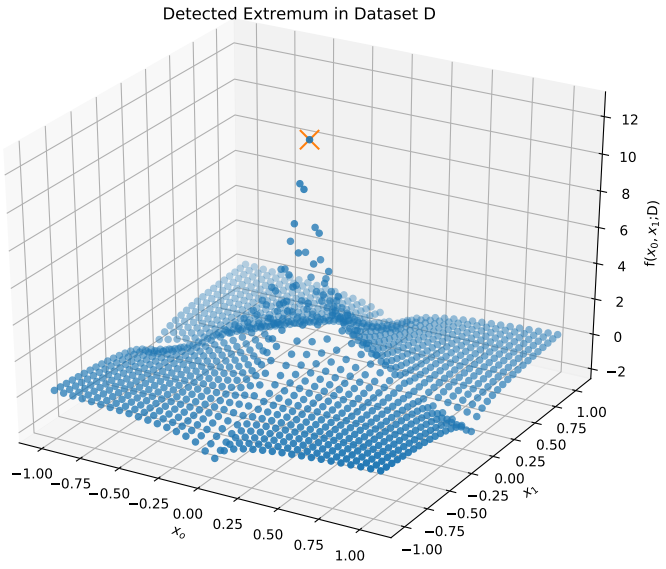
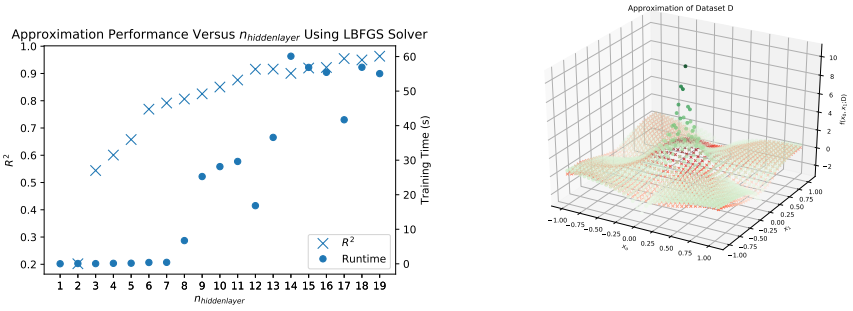



Figure 4.4: Detected extrema in logarithmic function (Eq. 4.5; a=1) data set. In $D$, there is one extrema ($M = 1$). The approximating function is Eq. 4.5 with a=1

The function is discontinuous at the origin and there is an obvious fold into the middle of each edge. The approximation captures the folds well per Fig. 4.5b, however, it struggles to approximate such a sharp peak. Using 15x more hidden layer neurons than what Zhang's theory proposes, the network is able to reach a near-perfect score.




(a) (b)

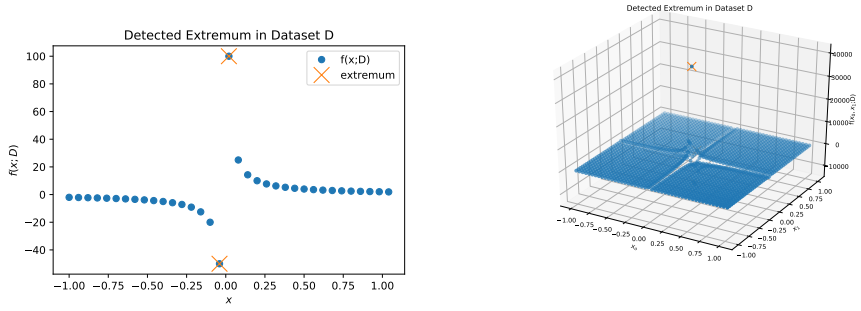
Figure 4.5: (a) Approximation performance in the form of $R^2$ and training time against $n_{l_1}$. (b) Overlay of the real set $D$ (circles) and the approximated values (crosses). In $D$, there is one extrema ($M = 1$). The approximated function is Eq. 4.5 with $a = 1$

Zhang's theory does not claim to approximate the entire data set perfectly, just the extrema. In turn, it makes sense that models made up of functions not-geometrically alike the sigmoidal activation neurons require more basis functions than Zhang's theory proposes.

Lastly, a much sharper peak was designed using a product of fractions.

$$f(x_i, x_{i+1}, ..., x_n) = \prod_{i=0}^{n} \frac{a}{x_i}. \tag{4.6}$$

The data is so discontinuous in Fig. 4.6b that the network was not able to train beyond $R^2 > 0.1$ with a hidden layer size within 500 % of M.




(a) (b)

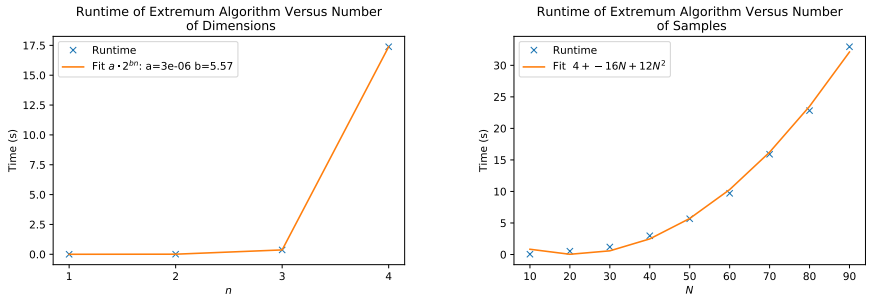
Figure 4.6: (a) Extrema detection of 2D reciprocal function. In $D$, there are two extrema ($M = 2$). (b) Extrema detection of 3D reciprocal function. In $D$, there are two extrema ($M = 2$). The approximated function is Eq. 4.6 with $a = 1$

### 4.1.4 Discussion

Based on the results from the logarithmic and reciprocal functions, if researchers are not receiving good results with hidden layer sizes close to the number of extrema in their data set, there may be a discontinuity that has to be accounted for by increasing the hidden layer size further.

The existance of discontinuities makes it difficult to confirm whether randomly searching using a heuristic like binary search or order of magnitude is slower than doing a linear search around the hidden layer size proposed by the EE theory. With this in mind, the extrema detection algorithm runtime becomes critical to whether or not Zhang's work will be used in further applications.

The rate of increase for the extrema detection algorithm is visualised in Fig. 4.7.

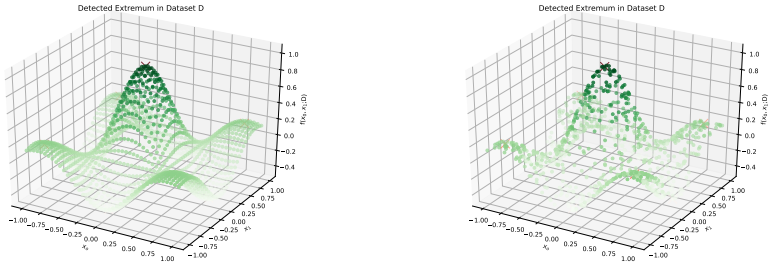



(a) (b)

Figure 4.7: Plot of extrema algorithm runtime against (a) 20 samples of an $n$-dimensional decaying sinewave function (b) $N$ samples of an 3D decaying sinewave function

The algorithm scales exponentially with $n$ and polynomially with $N$. The most effective way to keep costs low is to keep $n$ small.

If $n$ has already been reduced as much as possible to shrink algorithm cost, the next best option is to reduce $N$. One method to keep $N$ small is to use a stochastic approach (Fig. 4.8b) instead of using a uniform distribution (Fig. 4.8a). While a uniform distribution forces $N = (N_i)^n$, a stochastic distribution allocates the exact number of samples requested by the user. The random nature of where the mapping is sampled provides an alternative method to locate extrema.




(a) (b)

Figure 4.8: (a) Detected extrema in a uniformly distributed data set. (b) Detected extrema in a stochastically distributed data set

The cost of the stochastic approach can be shown as

$$O(2^n \cdot K), \tag{4.7}$$

where the K-multiplier accounts for the cost of the proximity and direction sub-algorithm. In its exhaustive form, the algorithm must take $N$ passes and compare each point to its $N-1$ neighbours using the $\mathcal{L}^2$ norm that performs $n$-time operations. In summary, $K = nN(N-1)$ which makes the total cost

$$O(nN(N-1) \cdot 2^n). \tag{4.8}$$

When comparing 4.1 and 4.8, the stochastic algorithm is less expensive if the following inequality is satisfied

$$nN_{stochastic}(N_{stochastic} - 1) < N_{uniform\ distribution}. \tag{4.9}$$

The uniform (Eq. 4.1) and stochastic (Eq. 4.8) time complexities demonstrate that at best, for exhaustive algorithms, time scales in a polynomial manner with respect to $N$. In contrast, backpropagation scales linearly with respect to $N$. While there are many other factors involved in searches, such as number of models trained and number of training epochs per model, it

is reasonable to assume that as $N$ grows, the time advantage of starting a search at an analytically selected number of hidden layer neurons disappears. For function approximation, $N$ is expected to grow with the need for better approximations.

There are some technical challenges that increase the cost of the stochastic algorithm. To find the neighbours of a reference point, every other point $P \in D; P \neq reference$ must be evaluated for proximity and categorized by cell; this is the exhausted algorithm version. An example of cells are the four quadrants in a 2D cartesian plane. For reference points in $\mathbb{R}^n$, there are $2^n$ possible cells for $P$ to be placed in. Pseudo code for the algorithm is shown below.

```
def which_cell(point, reference):
    cell_string = ""
    for dim in range(len(reference)):
        if point[dim] > reference[dim]:
            cell_string += '0'
        elif point[dim] < reference[dim]:
            cell_string += '1'
    return int(cell_string, 2)
```

Organizing $P$ by distance/direction ensures that the algorithm finds the $2^n$ points that form a four sided polygon around the reference point.

Dependent sampling methods like the random walk metropolis algorithm (RW) [63] could be a sensible next step following uniform and stochastic sampling. RW would create a subset of samples with an inclination for samples from areas in the data with high variance. Borrowing Zhang's assumption, that extrema contain important information, this would ensure the model is trained on a more representative sample. The obvious drawback is the increased cost related to the RW algorithm.

Judging by the fitted curve in Fig 4.7a, applying the EE algorithm in a nine-feature, thirteen-label data set would take months, much longer than typical random searches. Therefore, the algorithm was not used in the fol-

lowing PF approximation experiments.

## 4.2 Function Approximation: Non-Linear Power Flow

### 4.2.1 Problem and Objective

PF simulations can be expensive. As shown back in Fig. 1.7, a simulation of 10 million nodes could take almost 2 days. In practice, omissions in the input profile, whether by mistake or intentionally, can lead to very costly re-simulations.

The objective of §4.2 is to demonstrate the PF approximating process. Ideally the model should meet IEEE simulation standards (taken from the test feeder website[2]).

> 4-Bus Test Feeder Cases: ... Since the problems are so small, very close agreement with the test feeder results is expected. A good match would have an error less than 0.05%

Assuming voltage $V$ and predicted voltage $\hat{V}$ are in per-unit, the following inequality should hold.

$$MAXAE(V(\mathbf{x}),\ \hat{V}(\mathbf{x}))_{\mathbf{x}\in\mathbb{R}^n} < 0.0005. \tag{4.10}$$

### 4.2.2 Introduction

The example topology used for PF approximation comes from the institute of electrical and electronics engineers (IEEE). Since these circuits were intended for evaluating new power-flow methods on multi-phase distribution system models [64], which PyPSA is not capable of, a "pseudo-IEEE" circuit was built. Some modifications include:

[2] http://sites.ieee.org/pes-testfeeders/resources/

1. Unbalanced three-phase loading was summed into single phases
2. Neutral line parameters were substituted for phase line parameters
3. Latitude and Longitude were approximated
4. Switches were approximated as short lines
5. No voltage regulator was used

While the PF results do not line up one-to-one, the overall trends in the results are consistent between original/modified networks.

First, the following topology is set up in PyPSA. The network diagram contains all of the node numbers referenced in this section. The network topology is determined by node-line/transformer-node connections as well as generation placement.

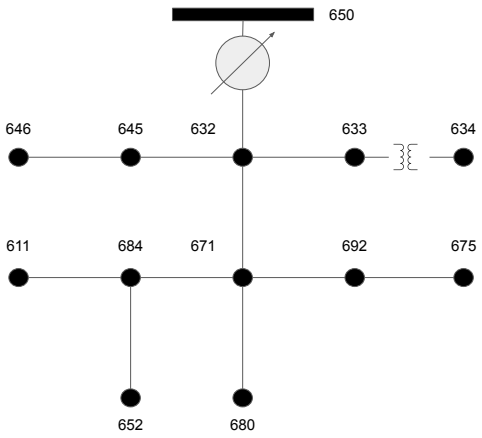



Figure 4.9: IEEE 13 Node Test Feeder. Copy of original work in [66]

The second step is setting up the inputs (snapshots of node loading) to the simulation. The load profiles, all similar to Fig. 4.10, generated by sampling $N$ points from a uniform distribution bounded by [0, 0.5], were imported into the simulation.

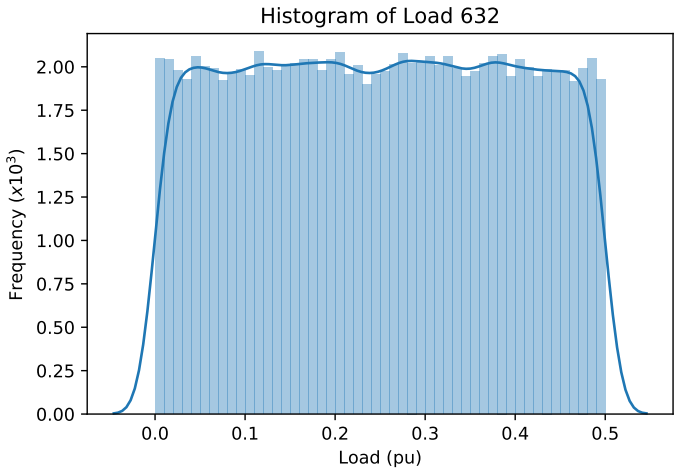



Figure 4.10: Load distribution at node 632 for PF simulation

Next, for the third step, the power flow simulation is run and the output is collected. The outputs from this simulation include snapshots of line loading, real/reactive voltage magnitude and the voltage phase angles at each bus. Each voltage profile looked similar to Fig 4.11.

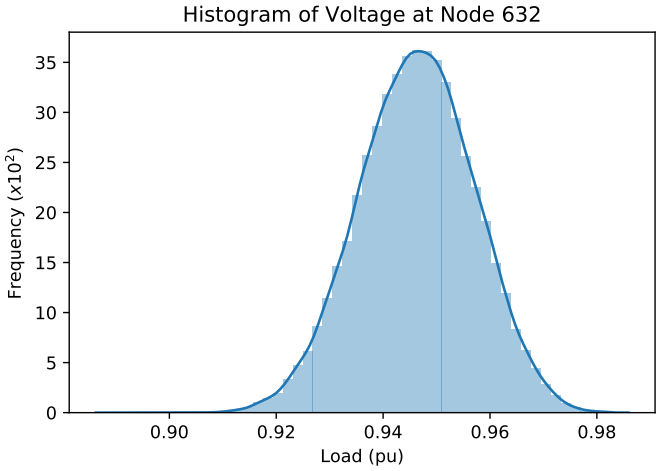



Figure 4.11: Voltage distribution at node 632 for PF simulation

### 4.2.3 Results

#### Data Analysis

First a correlation matrix is used to determine feature-to-feature relationships.

| | load-632 | load-634 | load-645 | load-646 | load-652 | load-671 | load-675 | load-692 | load-611 |
|---|---|---|---|---|---|---|---|---|---|
| **load-632** | 1.00 | 0.00 | -0.01 | 0.00 | 0.00 | 0.00 | -0.01 | 0.00 | 0.00 |
| **load-634** | | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | -0.01 | 0.00 | 0.00 |
| **load-645** | | | 1.00 | 0.00 | -0.01 | 0.00 | -0.01 | 0.00 | 0.00 |
| **load-646** | | | | 1.00 | 0.01 | 0.00 | 0.00 | 0.00 | 0.00 |
| **load-652** | | | | | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **load-671** | | | | | | 1.00 | -0.01 | 0.00 | 0.00 |
| **load-675** | | | | | | | 1.00 | 0.00 | 0.00 |
| **load-692** | | | | | | | | 1.00 | 0.00 |
| **load-611** | | | | | | | | | 1.00 |

Table 4.2: Pearson's *r* correlation between loads in 13-node network

Looking at the non-diagonal elements in Table 4.2, all of the features are weakly related and should be kept so long as they relate to a target. Since the feature sets were created by randomly sampling uniform distributions, this was expected. Next a plot of the voltage/load data is presented.

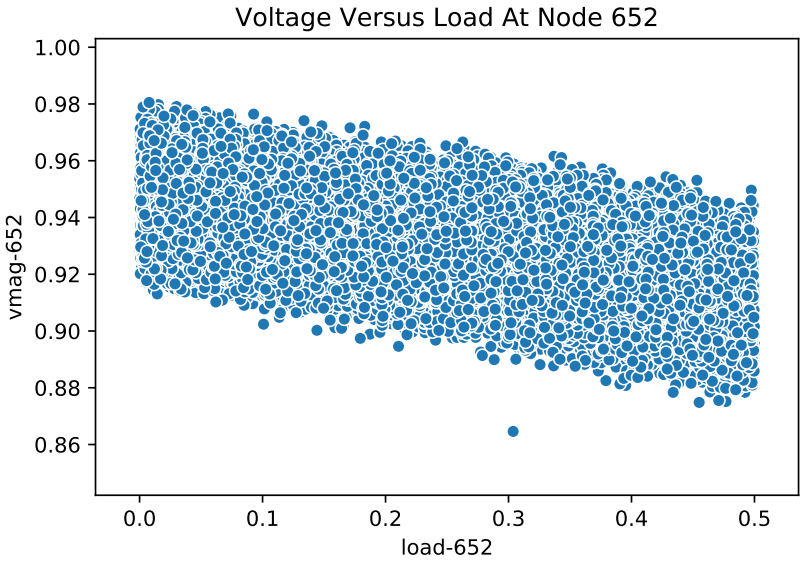



Figure 4.12: Plotting load versus voltage data at the most correlated node in the 13-node network

Pearson's $r$ is calculated on a few assumptions: homoscedasticity and normally distributed variables. It is obvious from Fig. 4.12 that neither condition is met; the spread is very heteroscedastic and neither variable is normally distributed. Despite the aforementioned drawbacks, there is clearly a linear trend and therefore $r$ may provide useful insight. To verify the results, Kendall's $\tau$, a non-parametric alternative that is only conditional on the data being ordinal, is used. If both $r$ and $\tau$ show the same trends, more confidence can be placed in any derived inferences.

| | load-611 | load-632 | load-634 | load-645 | load-646 | load-652 | load-671 | load-675 | load-692 | row average |
|---|---|---|---|---|---|---|---|---|---|---|
| **vmag-611** | -0.60 | -0.16 | -0.16 | -0.16 | -0.17 | -0.46 | -0.32 | -0.31 | -0.32 | -0.29 |
| **vmag-632** | -0.34 | -0.32 | -0.32 | -0.32 | -0.33 | -0.34 | -0.33 | -0.32 | -0.33 | -0.33 |
| **vmag-633** | -0.31 | -0.29 | -0.51 | -0.29 | -0.30 | -0.31 | -0.30 | -0.29 | -0.30 | -0.32 |
| **vmag-634** | -0.31 | -0.29 | -0.51 | -0.29 | -0.30 | -0.31 | -0.30 | -0.29 | -0.30 | -0.32 |
| **vmag-645** | -0.24 | -0.22 | -0.22 | -0.55 | -0.56 | -0.24 | -0.23 | -0.22 | -0.23 | -0.30 |
| **vmag-646** | -0.21 | -0.20 | -0.20 | -0.49 | -0.67 | -0.21 | -0.20 | -0.20 | -0.21 | -0.29 |
| **vmag-652** | -0.41 | -0.14 | -0.14 | -0.14 | -0.15 | -0.70 | -0.28 | -0.28 | -0.28 | -0.28 |
| **vmag-671** | -0.41 | -0.20 | -0.20 | -0.20 | -0.21 | -0.41 | -0.40 | -0.40 | -0.40 | -0.31 |
| **vmag-675** | -0.39 | -0.19 | -0.19 | -0.19 | -0.20 | -0.39 | -0.38 | -0.48 | -0.39 | -0.31 |
| **vmag-680** | -0.41 | -0.20 | -0.20 | -0.20 | -0.21 | -0.41 | -0.40 | -0.40 | -0.40 | -0.31 |
| **vmag-684** | -0.50 | -0.17 | -0.17 | -0.17 | -0.18 | -0.50 | -0.34 | -0.34 | -0.35 | -0.30 |
| **vmag-692** | -0.41 | -0.20 | -0.20 | -0.20 | -0.21 | -0.41 | -0.40 | -0.40 | -0.40 | -0.31 |
| **column average** | -0.38 | -0.21 | -0.25 | -0.26 | -0.29 | -0.39 | -0.32 | -0.33 | -0.33 | |

Table 4.3: Pearson's $r$ correlation between load and voltage magnitude

| | load-611 | load-632 | load-634 | load-645 | load-646 | load-652 | load-671 | load-675 | load-692 | row average |
|---|---|---|---|---|---|---|---|---|---|---|
| **vmag-611** | -0.42 | -0.10 | -0.10 | -0.10 | -0.11 | -0.31 | -0.21 | -0.20 | -0.21 | -0.20 |
| **vmag-632** | -0.23 | -0.21 | -0.21 | -0.21 | -0.22 | -0.22 | -0.22 | -0.21 | -0.22 | -0.22 |
| **vmag-633** | -0.20 | -0.19 | -0.35 | -0.19 | -0.20 | -0.20 | -0.20 | -0.19 | -0.20 | -0.21 |
| **vmag-634** | -0.20 | -0.19 | -0.35 | -0.19 | -0.20 | -0.20 | -0.20 | -0.19 | -0.20 | -0.21 |
| **vmag-645** | -0.15 | -0.14 | -0.14 | -0.37 | -0.38 | -0.15 | -0.15 | -0.14 | -0.15 | -0.20 |
| **vmag-646** | -0.14 | -0.13 | -0.13 | -0.33 | -0.48 | -0.14 | -0.13 | -0.13 | -0.13 | -0.19 |
| **vmag-652** | -0.27 | -0.09 | -0.09 | -0.09 | -0.10 | -0.50 | -0.18 | -0.18 | -0.18 | -0.19 |
| **vmag-671** | -0.27 | -0.13 | -0.13 | -0.13 | -0.14 | -0.27 | -0.27 | -0.26 | -0.27 | -0.21 |
| **vmag-675** | -0.26 | -0.12 | -0.12 | -0.12 | -0.13 | -0.26 | -0.25 | -0.32 | -0.26 | -0.21 |
| **vmag-680** | -0.27 | -0.13 | -0.13 | -0.13 | -0.14 | -0.27 | -0.27 | -0.26 | -0.27 | -0.21 |
| **vmag-684** | -0.34 | -0.11 | -0.11 | -0.11 | -0.12 | -0.34 | -0.23 | -0.22 | -0.23 | -0.20 |
| **vmag-692** | -0.27 | -0.13 | -0.13 | -0.13 | -0.14 | -0.27 | -0.27 | -0.26 | -0.27 | -0.21 |
| **column average** | -0.25 | -0.14 | -0.17 | -0.17 | -0.20 | -0.26 | -0.21 | -0.22 | -0.22 | -0.20 |

Table 4.4: Kendall's $\tau$ correlation between load and voltage magnitude

Both correlation methods produce similar trends. Node 650, the slack bus, has a constant voltage of 1.0 and therefore $r$ and $\tau$ cannot be calculated (since constants have no standard deviation/all ranks are tied respectively). The load/voltage profiles at each node are most strongly self-correlated. On average, it appears that a few loads, specifically node 652 and 611 are more correlated than the others. This is confirmed using permutation importance.

| Weight | Feature |
|---|---|
| $0.5001 \pm 0.0455$ | load-652 |
| $0.3612 \pm 0.0254$ | load-611 |
| $0.1867 \pm 0.0200$ | load-671 |
| $0.1355 \pm 0.0120$ | load-646 |
| $0.1347 \pm 0.0103$ | load-675 |
| $0.1344 \pm 0.0107$ | load-692 |
| $0.1052 \pm 0.0035$ | load-645 |
| $0.0465 \pm 0.0021$ | load-631 |
| $0.0307 \pm 0.0023$ | load-632 |

Table 4.5: Results from permutation importance algorithm run on RF model using data from modified IEEE 13 network

**Training**

All models were trained using *sklearn* Python modules [43], and $k$-fold cross validation was used to ensure trained models were data-agnostic. In §4.2.3 tables, MEANAE is shorthanded as mae.

First, an RF model was trained using the following hyperparameters.

| Parameter | Default Value |
|---|---|
| Number of Estimators | 100 |
| Criterion | mse |
| Max Depth | None |

Table 4.6: RF training parameters

The model's $R^2$ results scale with $N$, and at 100000 samples the RF outperforms the linear regression model. There is a problem with the MAXAE result; it is two orders of magnitude too high.

| N_samples | 100 | 1000 | 10000 | 100000 |
|---|---|---|---|---|
| rf_fit_time | 0.1381 | 0.7730 | 9.7295 | 131.2990 |
| rf_score_time | 0.0272 | 0.0463 | 0.4541 | 6.0899 |
| rf_test_r2 | 0.4721 | 0.7134 | 0.8193 | 0.9233 |
| rf_test_rmse | 0.0061 | 0.0045 | 0.0033 | 0.0020 |
| rf_test_mae | 0.0047 | 0.0034 | 0.0025 | 0.0013 |
| rf_test_maxae | 0.0169 | 0.0177 | 0.0170 | 0.0210 |
| linear_fit_time | 0.0058 | 0.0019 | 0.0089 | 0.1492 |
| linear_score_time | 0.0012 | 0.0023 | 0.0112 | 0.1944 |
| linear_test_r2 | 0.8608 | 0.9032 | 0.9053 | 0.9058 |
| linear_test_rmse | 0.0015 | 0.0014 | 0.0014 | 0.0014 |
| linear_test_mae | 0.0011 | 0.0011 | 0.0011 | 0.0011 |
| linear_test_maxae | 0.0049 | 0.0051 | 0.0065 | 0.0075 |

Table 4.7: Baseline RF results varying $N$

Both the correlation and permutation importance matrices suggest most nodes contain valuable information for the model. Using PCA, the nine-feature data set was reduced to 6, 7 and 8 features. While the time to train the model reduced by 27%, 15% and 8%, the score also decreased 12%, 9% and 4% respectively. Depending on the application, this trade off may be acceptable for the user.

| n_features | 6 | 7 | 8 |
|---|---|---|---|
| rf_fit_time | 96.8466 | 111.7810 | 121.0650 |
| rf_score_time | 5.2078 | 5.4140 | 5.4762 |
| rf_test_r2 | 0.8141 | 0.8366 | 0.8807 |
| rf_test_rmse | 0.0041 | 0.0037 | 0.0028 |
| rf_test_mae | 0.0026 | 0.0024 | 0.0018 |
| rf_test_maxae | 0.0395 | 0.0311 | 0.0207 |
| linear_fit_time | 0.1376 | 0.1096 | 0.1093 |
| linear_score_time | 0.2050 | 0.1995 | 0.2112 |
| linear_test_r2 | 0.9058 | 0.9058 | 0.9058 |
| linear_test_rmse | 0.0014 | 0.0014 | 0.0014 |
| linear_test_mae | 0.0011 | 0.0011 | 0.0011 |
| linear_test_maxae | 0.0084 | 0.0083 | 0.0080 |

Table 4.8: RF results with PCA reduced features. $N$ = 1e5

The slack bus was then removed and the $R^2$ results increased by roughly 4% over the best current score (in Table 4.7). Even with the 27% improvement in MAXAE, it is still two orders of magnitude above the desired benchmark.

| **n_samples** | **100** | **1000** | **10000** | **100000** |
|---|---|---|---|---|
| rf_fit_time | 0.1230 | 0.7485 | 9.2767 | 123.6820 |
| rf_score_time | 0.0278 | 0.0557 | 0.4026 | 5.0141 |
| rf_test_r2 | 0.5929 | 0.7792 | 0.8917 | 0.9619 |
| rf_test_rmse | 0.0066 | 0.0048 | 0.0034 | 0.0020 |
| rf_test_mae | 0.0054 | 0.0037 | 0.0026 | 0.0014 |
| rf_test_maxae | 0.0169 | 0.0184 | 0.0161 | 0.0152 |
| linear_fit_time | 0.0026 | 0.0014 | 0.0083 | 0.1391 |
| linear_score_time | 0.0013 | 0.0022 | 0.0132 | 0.1969 |
| linear_test_r2 | 0.9806 | 0.9806 | 0.9818 | 0.9813 |
| linear_test_rmse | 0.0015 | 0.0014 | 0.0014 | 0.0015 |
| linear_test_mae | 0.0012 | 0.0011 | 0.0011 | 0.0011 |
| linear_test_maxae | 0.0046 | 0.0059 | 0.0074 | 0.0078 |

Table 4.9: RF results with slack bus (node 650) removed from label set

A random search was run to find the optimal number of estimators and max depth. The search was conclusive in that the more estimators and more depth (test included up to 100 in each) the better the result.

All of the linear fit results necessary for comparison have been shown. Next, an SVR model was trained using the following hyperparameters.

| Parameter | Default Value |
|---|---|
| Kernel | rbf |
| $\gamma$ | $\frac{1}{n_{features} \cdot \sigma(input)}$ |
| C | 1.0 |
| $\epsilon$ | 0.0002 |

Table 4.10: SVR training parameters. $\sigma$: variance

The baseline results show SVR requires much less data than RF to achieve better scores. At $N = 1000$, the SVR model trains 83% faster than its RF

counterpart in Table 4.7 and receives a 67% better MAXAE score.

| **N samples** | **10** | **100** | **1000** |
|---|---|---|---|
| svr_fit_time | 0.0006 | 0.001 | 0.13 |
| svr_score_time | 0.001 | 0.002 | 0.03 |
| svr_test_r2 | -65 | 0.74 | 0.88 |
| svr_test_rmse | 0.011 | 0.0036 | 0.0020 |
| svr_test_mae | 0.010 | 0.0027 | 0.0016 |
| svr_test_maxae | 0.014 | 0.0093 | 0.0069 |

Table 4.11: SVR results averaged over all labels

When the slack bus is removed, $R^2$ results improve by 9%. The MAXAE is 51% better than the equivalent RF score and 1% over the best linear regression score. However, losing the slack bus resulted in a 7% drop from the best SVR MAXAE.

| **N samples** | **10** | **100** | **1000** |
|---|---|---|---|
| svr_fit_time | 0.0006 | 0.001 | 0.14 |
| svr_score_time | 0.002 | 0.002 | 0.03 |
| svr_test_r2 | -70 | 0.82 | 0.96 |
| svr_test_rmse | 0.012 | 0.0038 | 0.0022 |
| svr_test_mae | 0.011 | 0.0029 | 0.0017 |
| svr_test_maxae | 0.015 | 0.0100 | 0.0074 |

Table 4.12: SVR results with slack bus (node 650) removed from label set

A randomised search was run with $N = 10000$ and the $mode(R^2) = 0.98$. Unfortunately the MAXAE was not captured in this experiment, however the RMSE values are 35% better than the previous experiment. A simple ratio estimates the *mode*(MAXAE) values in Table 4.13 at 0.0047.

| fit (s) | kernel | gamma | epsilon | C | r2 | rmse | std |
|---|---|---|---|---|---|---|---|
| 42 | linear | scale | 0.0009 | 4.7 | 0.98 | 0.00142 | 0.00001 |
| 30 | linear | auto | 0.0011 | 4.1 | 0.98 | 0.00142 | 0.00001 |
| 3 | rbf | auto | 0.0006 | 0.5 | 0.98 | 0.00143 | 0.00000 |
| 2 | linear | scale | 0.0016 | 0.3 | 0.98 | 0.00143 | 0.00001 |
| 6 | rbf | auto | 0.0012 | 3.2 | 0.98 | 0.00143 | 0.00001 |
| 3 | linear | auto | 0.0015 | 0.4 | 0.98 | 0.00143 | 0.00001 |
| 3 | rbf | auto | 0.0003 | 0.2 | 0.98 | 0.00143 | 0.00000 |
| 4812 | poly | scale | 0.0017 | 4.4 | 0.94 | 0.00276 | 0.00009 |
| 3717 | poly | scale | 0.0009 | 1.9 | 0.93 | 0.00277 | 0.00009 |
| 5 | sigmoid | auto | 0.0009 | 4.7 | -1.05E+01 | 0.03685 | 0.00121 |

Table 4.13: SVR grid search with $N = 1e4$. Table has been trimmed to remove similar parameters for readability

Lastly, ANNs were trained with the following hyperparameters.

| Hyperparameter | Default Value |
|---|---|
| Solver | lbfgs |
| L2 Penalty | 1e-6 |
| Batch Size | min(200, N) |
| Initial Learning Rate | 1e-6 |
| Maximum Iterations | 3000 |
| Tolerance | 1e-9 |
| Number of Iterations With No Change (Saturation) | 100 |

Table 4.14: ANN training parameters

Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) is selected as the default solver because it is the least likely to get stuck in a local minimum or on a plateau[3]. For larger data sets ADAM is likely the best solver because the number of computations scales less than LBFGS with data/features, and adjusts parameters in-flight making it usually faster than SGD. It also performs comparably to other known stochastic optimization methods[4] [26]. However, Keskar argues that SGD

[3]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
[4]http://ruder.io/optimizing-gradient-descent/

outperforms ADAM in later stages of training, which would lead to better approximations [41].

Training with small data sets typically lead to unusable networks ($R^2$ scores below 0) much like the scores for $N = 100, 1000$ below.

| N samples | 100 | 1000 | 10000 | 100000 |
|---|---|---|---|---|
| dnn_fit_time | 0.8411 | 2.9870 | 26.1736 | 123.0660 |
| dnn_score_time | 0.0017 | 0.0029 | 0.0190 | 0.2396 |
| dnn_test_r2 | -13371.8000 | -7001.4600 | 0.2571 | 0.9687 |
| dnn_test_rmse | 0.9304 | 0.7839 | 0.0094 | 0.0018 |
| dnn_test_mae | 0.8868 | 0.7340 | 0.0073 | 0.0014 |
| dnn_test_maxae | 1.3626 | 1.2036 | 0.0412 | 0.0116 |

Table 4.15: ANN results varying $N$ with slack bus (node 650) removed from label set. Each ANN model had two 10-neuron hidden layers

While 0.97 is among the best $R^2$ score a non-linear regression network achieved, the MAXAE value is 68% worse than the best confirmed SVR score. A random search was run to find trends in solver, hidden layer size and activation function categories.

| fit (s) | solver | h_l_s | activation | r2 | rmse | std | mae |
|---|---|---|---|---|---|---|---|
| 458 | adam | (5,) | relu | 0.90 | 0.0033 | 0.0000 | 0.0026 |
| 4 | lbfgs | (7, 7) | relu | 0.87 | 0.0038 | 0.0002 | 0.0028 |
| 5 | lbfgs | (7, 6) | relu | 0.86 | 0.0038 | 0.0005 | 0.0028 |
| 4 | lbfgs | (7, 8) | log. | 0.82 | 0.0046 | 0.0010 | 0.0034 |
| 13013 | sgd | (6, 6) | relu | 0.72 | 0.0059 | 0.0003 | 0.0045 |
| 4 | lbfgs | (5, 9) | tanh | 0.68 | 0.0057 | 0.0022 | 0.0044 |
| 3 | lbfgs | (7, 6) | relu | 0.59 | 0.0060 | 0.0034 | 0.0046 |
| 2 | lbfgs | (7, 9) | log. | 0.34 | 0.0082 | 0.0030 | 0.0064 |
| 763 | adam | (6,) | log. | 0.19 | 0.0097 | 0.0001 | 0.0076 |
| 10248 | sgd | (6,) | log. | 0.14 | 0.0101 | 0.0000 | 0.0079 |

Table 4.16: ANN grid search with $N = 1e5$. "h_l_s": hidden layer size

The ReLU activation function typically outperformed the logistic sigmoid function. This is corroborated by the fact that linear regression models have been out-performing most models, and ReLU is a linear activation

function when the inputs are positive. Another interesting observation is that the LBFGS solver is faster than Adam, even with $N = 100000$. Often, root-finding methods become costly with large samples, but in this scenario perhaps there were few local minima for the method to get caught in. Lastly, SGD had very long run times. This is likely due to plateaus in the solution space; since SGD had a constant, small learning rate, it required many iterations to reach optimum. This could likely be improved by using an adaptive learning rate, but based on Adam's performance it is unlikely that would outperform LBFGS.

### 4.2.4 Discussion

This research concludes that the model with the combination of fastest / most accurate approximation is an SVR model with an RBF kernel. The MAXAE score of 0.0069 outperforms the next best, a linear regression model, by 8%. Unfortunately, the MAXAE scores are still 1280% off the IEEE recommended tolerance. Based on the MEANAE of 0.0016 and a standard deviation an order of magnitude below (a common trend seen in all models - omitted for brevity), only an infinitesimal number of points would stay in-bounds. This work is applicable to those interested in observing trends in PF data and not for precision use.

RF models trained in a much more consistent time than ANNs and had similar results. The results show that the RF score scaled quickly with more samples. With the knowledge that RF performs better, in general, on networks with more features, it will likely outperform SVRs score-wise and ANNs time-wise on topologically larger, more sample dense PF problems.

Fit time for SVR models scales very fast. In Table 4.12, the fitting time grew 67% and 139% between $N = 10/100$ and $N = 100/1000$ respectively. In the randomised search, linear models (equivalent to linear regression) and rbf models performed the same, with polynomial and sigmoid kernel models significantly behind.

ANNs were the most difficult to train. They failed often, reaching very large, negative scores. This was 3x less likely when using LBFGS over

Adam. Increasing sample size was the most obvious way to improve results. The randomised search in Table 4.16 showed that while Adam could achieve the best scores ($R^2 = 0.9$, an 3.4% improvement over the next best), it took 113x more time than the next best model which was trained using LBFGS.

In Wolport's famous "No Free Lunch" paper [68], he describes that unless you make assumptions about the data, it is impossible to select one model over another a priori. Based on the work in this thesis and by our group in [7], the input/output correspondence for PF simulations appears to be very linear even in high dimension problems. Results show it is best to use an SVR model with an RBF kernel for approximating the network.

# Chapter 5

# Simulation Approximation Methodology

After discerning the best model to approximate a PF network with, this chapter describes a high-level architecture of how to insert the function approximation process into a simulation program. First, proper data management methodology is introduced, which provides theory on how to select the input domain for a simulation.

## 5.1 Data Selection and Preparation

This thesis demonstrated *batch*, *model*-based learning (as opposed to any other combination of *batch/online, model/instance*) because our target function is stationary and a parameterized solution was deployed [52]. In other words, data was used to train the parameters of a model such that it is representative of the simulation where the data was sourced from.

Data sourcing and handling is an extremely important piece of the training process. One often-associated idiom is "Garbage in, garbage out." To provide the best data to the model, consideration should be given to:

1. Amplitude
2. Feature space coverage

Amplitude is important because larger steps in solvers like gradient descent skew towards larger values. Normalising the data set makes the impact of each member during training more equivalent. This ensures the model is more representative of sample density, not sample magnitude. One must also consider sampling bias. Models trained on imbalanced data (majority/minority classes) will be more sensitive to the majority class. For regression, this would come across as a skew towards majority subsets in feature space.

Feature space coverage is important because a model can only approximate what it is exposed to. A uniform distribution guarantees feature space will be covered within a domain.

$$\boldsymbol{x} \in R^n : \boldsymbol{x}^1 = \boldsymbol{x}^2 = ... = \boldsymbol{x}^n, \tag{5.1}$$

where $N_{uniform} = \sum_{i=0}^{n} len(\boldsymbol{x}^i)$. It may, however, be more effective to use a stochastic feature space distribution where $N_{stochastic} \ll N_{uniform}$,

$$\boldsymbol{x} \in R^n : \boldsymbol{x}^i = random[\mathbb{D}_{start}, \mathbb{D}_{stop}]. \tag{5.2}$$

Where $\mathbb{D}$ is the domain. This is akin to typical Boltzmann machine implementations [48] [53]. The randomness can produce a representative sample of the data set with smaller input cardinality than its uniform equivalent.

In summary, the larger and more dense your domain is, the more detail the approximation can capture. Before training a model to approximate a data set, input equilisation techniques should be used to reduce bias of any kind. The aforementioned information should be applied when selecting input data for the file manager (FM) system proposed below.

## 5.2 File Manager

After discerning the best model to approximate a PF network with, this chapter briefly describes a high-level architecture of how to insert the function approximation process into a simulation program.

Most simulation programs follow the same pattern.

1. A problem description containing the topological mapping between atomic units is drafted using a graphical user interface or imported as a text file. This creates the underlying model

2. Input data representing the specific behaviour of each unit is imported into the model as a set of sample vectors

3. The simulation is run, feeding each sample vector to the ensemble of units one snapshot at a time

Fig. 5.1 interrupts this process by inserting a FM after the initialisation step. The FM reads the problem description and determines whether or not it has seen it before. If it has not, it runs the solver as the simulation would normally and uses the inputs from step 2, as well as the outputs from the solver in step 3 to create an approximation. The approximation is then appended to the problem description and saved to a folder. The next time a problem description is read, the FM searches the folder for that description and if it exists, the simulation feeds the inputs into the approximation to get the outputs instead of using the solver.

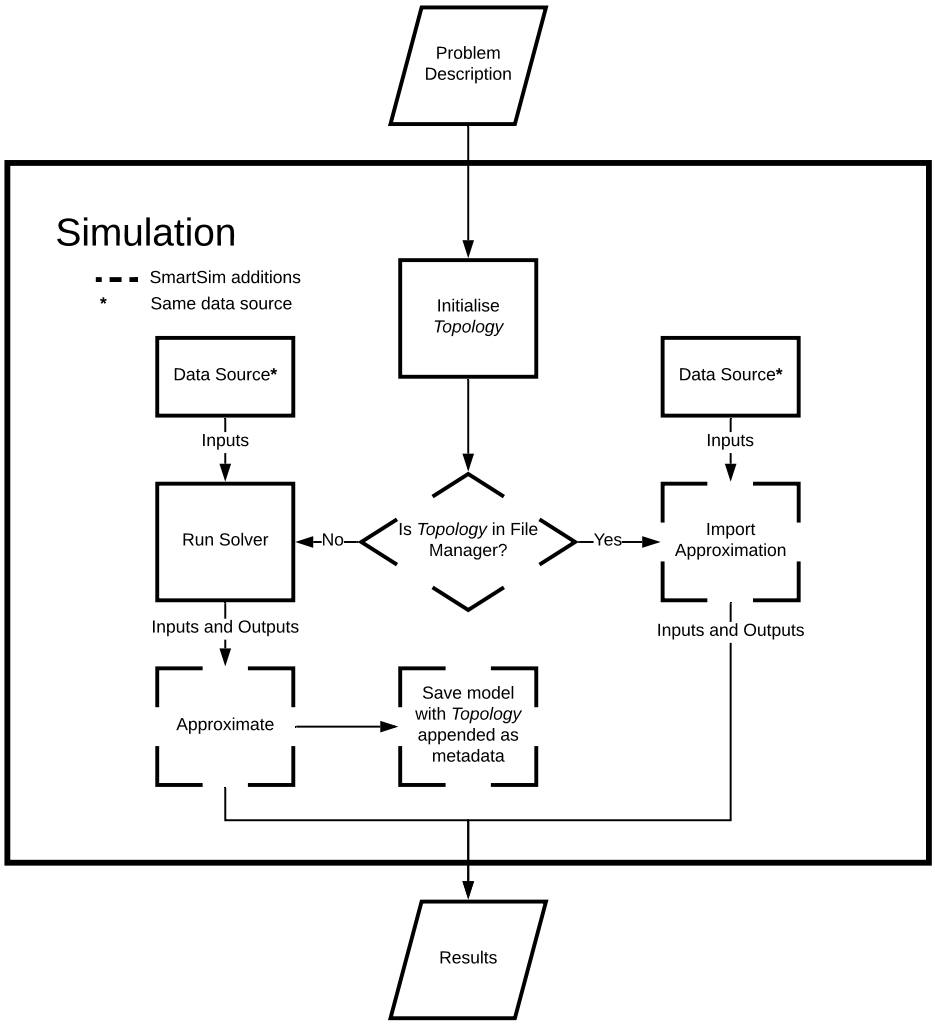



Figure 5.1: Flow chart of simulation program with embedded file manager and automatic approximating modules

After simulations are approximated, their metadata (simulation context, approximation parameters) can be stored into a directory by a file manager for future reference. When initialising any simulation, the file manager first searches the directory for a simulation context matching the reference context. If the same context is found, the approximation is used

instead of re-running the iterative simulation solver.

The slowest method of comparing simulation context would be an exhaustive search. The program would open each file, $F_i : i_{max}$ = total number of files in the directory and compare the metadata with reference context, $R$, line for line. This has a time complexity of $O(R_{n\_lines} \cdot i_{max})$.

A faster approach would be to cross-reference keyed values in each file to see if they match and assume that if they do, the files are the same. The file manager checks a random keyed value. If a key, $K$, does not exist or the value, $V$, is wrong, the contexts are not the same, $F_i \neq R$. The probability that the value is the same, depends on the number of possible variations for that value. Assuming for $V$, there are $C$ possible scalars randomly selected from a uniform distribution,

$$P(V_{F_i,K} == V_{R,K}) = \frac{1}{C}. \tag{5.3}$$

To decrease the probability of a false positive context equivalency, the number of values tested, $j$, can be increased. The advantages of this method come with its simple implementation and smaller time complexity $O(j \cdot i_{max}) < O(R_{n\_lines} \cdot i_{max})$.

The optimal method is string hashing, which is used by Python for dictionary objects, for example. Using hashing methods like Division, Multiplication, Folding, Random Number Generator [69] [70], or a hash function like Single Hash [71] and assuming each context is sorted by the same keys, a unique value can be assigned to each context for $O(B)$ lookup. For a simple hashing example, we could replace the first four characters, *"sim*, of the sample PyPSA context file,

```
"simulationtype": "PyPSA",
"connections": [[0,1,0],[1,0,1],[0,1,0]],
"start_datetime": "2016-04-01 0:0:0",
"profiles": [{"load":true,"generation":true,"storage":false},
{"load":true,"generation":false,"storage":false},
{"load":true,"generation":false,"storage":false}],
"profile_path": "./",
"study": {"description":"binary2-m4y16-super-load"},
"lookup_table": false
```

with its ASCII-decimal equivalent,

$$\text{"sim} \rightarrow 34 + 115 + 105 + 109.$$

This sum is divided by the desired size of the hash table and the given result would be the entries index. This strategy is used often, for example, in Linux where the SHA-256 key is offered so users can check if a download is corrupt.

A key improvement to the proposed simulation program would be to accept both exact and approximate problem descriptions during the matching process. For example, large PF networks with multiple slack buses may contain sub-topologies that adequately mimic the topologies in standalone networks.

The cost savings associated with this model are broken down into two categories:

1. $time_{model-training} \gg time_{model-evaluating}$
2. $time_{model-training} \ll time_{model-evaluating}$.

Option 1 would occur if the model is expensive to train and evaluated sparsely. In this case,

$$speed\ increase\ \alpha\ n_{simulations}. \tag{5.4}$$

In other words the speed increase will be proportional to the number of times that problem description is used. Option 2 occurs if the model is heavily evaluated, such as in the event that a computer relies on the simulation for guiding another process. In case 2,

$$speed\ increase\ \sim\ \frac{time_{one-simulation}}{time_{one-evaluation}} \tag{5.5}$$

A SVR example with $N = 10000$ shows the possible savings using the FM. In the event that one hundred simulations were run, the FM would provide answers 100x faster. If 1e8 simulations were run, the time advantage scales to 100 000x.

| Scenario | 1 Simulation (m) | 100 Simulations (m) | 1e8 Simulations (m) |
|---|---|---|---|
| | | $t_{train} \gg t_{evaluate}$ | $t_{train} \ll t_{evaluate}$ |
| Mapped | $t_{sim}$ = 420<br>$t_{train}$ = 14<br>$t_{evaluate}$ = 0.001 | $t_{controlled}$ = 4e4<br>$t_{approxs}$ = 4e2 | $t_{controlled}$ = 4e10<br>$t_{approx}$ = 1e5 |

Table 5.1: Runtime comparison of repeated simulations on regular simulation software versus a simulation software with an embedded file manager

# Chapter 6

# Conclusion

The major objective of this thesis was to prove it was possible to approximate non-linear, power flow simulations. Ideally, the maximum absolute error for the approximation would be below 0.05% per the recommendation from the Institute of Electrical and Electronics Engineers. Multiple experiments tested the regression ability of random forest, support vector regression, fully connected feedforward artificial neural network and linear regression models. Each model was able to eclipse the 2% maximum absolute error mark, with support vector regression producing the lowest error at 0.69%. The training procedure and models used in this thesis are best applied to problems where data trends are required, not precision.

The second goal was to provide an analytical method to select hidden layer width for a three-layer ANN. An extrema-based algorithm was written in Python and tested on sine, log, and reciprocal based n-dimensional data sets. It is shown for the continuous, sine-based function that setting the hidden layer size according to Zhang's rule produces extremely accurate approximations. Because the extrema detection algorithm runtime scales exponentially with $n$, the algorithm in its current exhaustive form is too costly for algorithms with more than five features and one thousand samples. It is faster to run a random, order-of-magnitude search.

The final objective in this thesis was to suggest a method to reduce the cost of repeating simulations on the same model. High-level simulation

program suggestions were presented, which by implementation would create a fully automated, time-reduced simulation software. This is achieved by inserting an approximation class based on the other research presented in this thesis and metadata storage into pre-existing simulation code. Time-saving theory was discussed and demonstrated in an example where, in the more likely scenario, savings up to 100x could be achieved.

The next steps to improving the extrema detection algorithm include optimising the polytope functions. For assurance that a point was an extremum, the current method assumes the point must have a neighbour in every cell that is not an extremum. In big-O notation, this is $O(2^n)$. More optimised polytope algorithms would only require n+1 cells to be filled and would be $O(n+1)$. It may also be useful to pivot to numerical algorithms such as simplex or steepest descent to find extremas as their time-complexity does not scale exponentially with number of features.

Other universal approximation methods, including radial basis functions or spline approximation, should be trialled. It may also be possible to use linear interpolation, however leading n-dimensional interpolation functions such as Numpy's LinearNDInterpolator crashed when presented with high-sample (>1000), high-dimension (>10) power flow data sets. It would also be interesting to use Jackson Polynomials to evaluate conversion rates. There are many conditions in which ANN training does not converge. For instance, training on periodic functions often leads to convergence in a local extremum, making a good approximation almost impossible. Many authors have proposed solutions for local-convergence scenarios, and this feature would significantly increase approximation robustness. It would also benefit the community to consider the nuances of the time-reducing simulation metadata to an open source cloud environment. What would the architecture look like? What type of security policies would have to be put in place?

# Bibliography


[1] B. Hidalgo and M. Goodman, "Multivariate or multivariable regression?" *American journal of public health*, vol. 103, no. 1, pp. 39–40, 01 2013.

[2] R. Lyster, "Smart grids: Opportunities for climate change mitigation and adaptation," *Monash UL Rev.*, vol. 36, p. 173, 2010.

[3] C. Shum, W. Lau, T. Mao, H. S. Chung, K. Tsang, N. C. Tse, and L. L. Lai, "Co-simulation of distributed smart grid software using direct-execution simulation," *IEEE Access*, vol. 6, pp. 20 531–20 544, 2018.

[4] S. Karnouskos and T. N. d. Holanda, "Simulation of a smart grid city with software agents," in *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, Nov 2009, pp. 424–429.

[5] P. Oliveira, T. Pinto, H. Morais, and Z. Vale, "Masgrip — a multi-agent smart grid simulation platform," in *2012 IEEE Power and Energy Society General Meeting*, July 2012, pp. 1–8.

[6] M. H. Carpenter, D. Gottlieb, S. Abarbanel, and W.-S. Don, "The theoretical accuracy of runge–kutta time discretizations for the initial boundary value problem: a study of the boundary error," *SIAM Journal on Scientific Computing*, vol. 16, no. 6, pp. 1241–1252, 1995.

[7] M. Bardwell and P. Musilek, "Enhancing power flow simulations using function mapping," in *2019 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2019, pp. 1–5.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[10] Z. Zainuddin and O. Pauline, "Function approximation using artificial neural networks," *WSEAS Transactions on Mathematics*, vol. 7, no. 6, pp. 333–338, 2008.

[11] X. M. Zhang, Y. Q. Chen, N. Ansari, and Y. Q. Shi, "Mini-max initialization for function approximation," *Neurocomputing*, vol. 57, pp. 389–409, 2004.

[12] R. Elkadiri, M. Sultan, A. M. Youssef, T. Elbayoumi, R. Chase, A. B. Bulkhi, and M. M. Al-Katheeri, "A remote sensing-based approach for debris-flow susceptibility assessment using artificial neural networks and logistic regression modeling," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 12, pp. 4818–4835, Dec 2014.

[13] G. May and A. El-Shahat, "Battery-degradation model based on the ann regression function for ev applications," in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, Oct 2017, pp. 1–3.

[14] I. Asiltürk and M. Çunkaş, "Modeling and prediction of surface roughness in turning operations using artificial neural network and multiple regression method," *Expert systems with applications*, vol. 38, no. 5, pp. 5826–5832, 2011.

[15] P. V. B. Reddy, C. H. R. V. Kumar, and K. H. Reddy, "Modeling of wire edm process using back propagation (bpn) and general regression neural networks (grnn)," in *Frontiers in Automobile and Mechanical Engineering -2010*, Nov 2010, pp. 317–321.

[16] H. Majumder and K. Maity, "Predictive analysis on responses in wedm of titanium grade 6 using general regression neural network (grnn) and multiple regression analysis (mra)," *Silicon*, pp. 1–14, 2018.

[17] L. Xia, J. Meng, R. Xu, B. Yan, and Y. Guo, "Modeling of 3-d vertical interconnect using support vector machine regression," *IEEE Microwave and Wireless Components Letters*, vol. 16, no. 12, pp. 639–641, Dec 2006.

[18] J. Meng, Y. Gao, and Y. Shi, "Support vector regression model for measuring the permittivity of asphalt concrete," *IEEE Microwave and Wireless Components Letters*, vol. 17, no. 12, pp. 819–821, Dec 2007.

[19] S. Zhang, M. Wang, P. Zheng, G. Qiao, F. Liu, and L. Gan, "An easy-to-implement hysteresis model identification method based on support vector regression," *IEEE Transactions on Magnetics*, vol. 53, no. 11, pp. 1–4, Nov 2017.

[20] Y. Zhang, Y. Du, F. Ling, S. Fang, and X. Li, "Example-based super-resolution land cover mapping using support vector regression," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 4, pp. 1271–1283, April 2014.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[22] M. A. Hannan, J. A. Ali, A. Mohamed, and M. N. Uddin, "A random forest regression based space vector pwm inverter controller for the induction motor drive," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 4, pp. 2689–2699, April 2017.

[23] I. Ghosal and G. Hooker, "Boosting random forests to reduce bias; one-step boosted forest and its variance estimate."

[24] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[25] L. Deng, J. Li, J. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero, "Recent advances in deep learning for speech research at microsoft," in *Proc. Speech and Signal Processing 2013 IEEE Int. Conf. Acoustics*, May 2013, pp. 8604–8608.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] J. Portsmouth, "Efficient barycentric point sampling on meshes," *arXiv preprint arXiv:1708.07559*, 2017.

[28] D.-J. van der Zee, "Model simplification in manufacturing simulation–review and framework," *Computers & Industrial Engineering*, vol. 127, pp. 1056–1067, 2019.

[29] N. Watson and J. Arrillaga, *Power Systems Electromagnetic Transients Simulation*. The Institution of Engineering and Technology, 2003.

[30] L. Guo, M. Wang, C. Ruan, T. Y. Lin, C. Yang, L. Wei, C. Geng, C. Xing, and Y. Xiao, "A cloud simulation based environment for multi-disciplinary collaborative simulation and optimization," in *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, April 2017, pp. 445–450.

[31] M. Zakarya and L. Gillam, "Modelling resource heterogeneities in cloud simulations and quantifying their accuracy," *Simulation Modelling Practice and Theory*, vol. 94, pp. 43 – 65, 2019.

[32] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary, "Scheduling critical channels in conservative parallel discrete event simulation," in *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99. (Cat. No.PR00155)*, May 1999, pp. 20–28.

[33] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[34] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, and H. Yang, "Large scale recurrent neural network on gpu," in *2014 International Joint Conference on Neural Networks (IJCNN)*, July 2014, pp. 4062–4069.

[35] Zhongwen Luo, Hongzhi Liu, and Xincai Wu, "Artificial neural network computation on graphic process unit," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 1, July 2005, pp. 622–626 vol. 1.

[36] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183–192, 1989.

[37] B. I. Hong and N. Hahm, "A note on neural network approximation with a sigmoidal function," *Applied Mathematical Sciences*, vol. 10, no. 42, pp. 2075–2085, 2016.

[38] H. N. Mhaskar, "Approximation properties of a multilayered feedforward artificial neural network," *Advances in Computational Mathematics*, vol. 1, no. 1, pp. 61–80, 1993.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification."

[40] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[41] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd."

[42] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, 2012. [Online]. Available: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[45] A. J. Smola and B. Schölkopf, "A tutorial on Support Vector Regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[46] Y. Li, J. Bai, Q. Tang, Y. Jiang, C. Li, and S. Xia, "Multinomial random forests: Fill the gap between theoretical consistency and empirical soundness."

[47] G. M. Phillips, *Interpolation and approximation by polynomials*. Springer Science & Business Media, 2003, vol. 14.

[48] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[49] Y. N. Dauphin and Y. Bengio, "Big neural networks waste capacity."

[50] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A modern take on the bias-variance tradeoff in neural networks."

[51] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010.

[52] A. Geron, *Hands-On Machine Learning With Scikit-Learn & Tensorflow. Concepts, Tools and Techniques to Build Intelligent Systems*, N. Tache, Ed. O'Reilly Media, Inc, 2017.

[53] G. Hinton, "Boltzmann Machines," Mar. 2007. [Online]. Available: https://www.cs.toronto.edu/~hinton/csc321/readings/boltz321.pdf

[54] S. J. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252–264, Mar. 1991.

[55] I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik, "What size test set gives good error rate estimates?" in *IEEE Trans PAMI*, 1996, pp. 52–64.

[56] I. Guyon, "A scaling law for the validation-set training-set size ratio," 1997.

[57] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[58] J. D. Gibbons and J. D. G. Fielden, *Nonparametric measures of association*. Sage, 1993, no. 91.

[59] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 04 2010. [Online]. Available: https://doi.org/10.1093/bioinformatics/btq134

[60] D. Clarke, "A convenient omitted variable bias formula for treatment effect models," *Economics Letters*, vol. 174, pp. 84–88, 2019.

[61] I. Jolliffe, *Principal Component Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_455

[62] K. Ohtani, "Bootstrapping r2 and adjusted r2 in regression analysis," *Economic Modelling*, vol. 17, no. 4, pp. 473–483, 2000.

[63] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335,

1995. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/00031305.1995.10476177

[64] K. P. Schneider, B. A. Mather, B. C. Pal, C. . Ten, G. J. Shirek, H. Zhu, J. C. Fuller, J. L. R. Pereira, L. F. Ochoa, L. R. de Araujo, R. C. Dugan, S. Matthias, S. Paudyal, T. E. McDermott, and W. Kersting, "Analytic considerations and design basis for the ieee distribution test feeders," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3181–3188, May 2018.

[65] "Ieee application guide for ieee std 1547(tm), ieee standard for interconnecting distributed resources with electric power systems," *IEEE Std 1547.2-2008*, pp. 1–217, April 2009.

[66] W. H. Kersting, "Radial distribution test feeders," *IEEE Transactions on Power Systems*, vol. 6, no. 3, pp. 975–985, Aug 1991.

[67] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning," Online, Jan. 2019.

[68] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[69] M. Singh and D. Garg, "Choosing best hashing strategies and hash functions," in *2009 IEEE International Advance Computing Conference*, March 2009, pp. 50–55.

[70] M. Ramakrishna and J. Zobel, "Performance in practice of string hashing functions," in *Database Systems For Advanced Applications' 97*. World Scientific, 1997, pp. 215–223.

[71] X. Gou, C. Zhao, T. Yang, L. Zou, Y. Zhou, Y. Yan, X. Li, and B. Cui, "Single hash: Use one hash function to build faster hash based data structures," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jan 2018, pp. 278–285.