NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

NEURAL NETWORK DISTRIBUTIONAL INITIAL CONDITION ROBUSTNESS IN POWER SYSTEMS

by

Philip B. Smith

June 2022

Thesis Advisor:	Wei Kang
Co-Advisor:	Thor Martinsen
Second Reader:	Anthony J. Gannon

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE NEURAL NETWORK DISTRIBUTIONAL INITIAL CONDITION ROBUSTNESS IN POWER SYSTEMS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Philip B. Smith	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

How can we measure and classify neural network robustness across differently distributed data to avoid misuse of machine learning tools? This thesis adopts several metrics to measure the initial condition robustness of feedforward neural networks, allowing the creators of such networks to measure and refine their robustness and performance. This could allow highly robust neural networks to be used reliably on untrained data distributions and prevent the use of less robust networks as a black box in a poor environment. We test this measurement of robustness on a series of differently sized neural networks trained to detect and classify microgrid power system faults, giving examples of both robust and nonrobust networks, along with suggestions on how to maximize robustness. The analysis reveals that collecting data from segments along trajectories enhances the robustness of neural networks. In such data sets, the distribution of data points is dominated by the dynamics of the system, not the initial state distribution.

14. SUBJECT TERMS neural network, robustness, microgrid, feedforward	**15. NUMBER OF PAGES** 67
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

NEURAL NETWORK DISTRIBUTIONAL INITIAL CONDITION ROBUSTNESS IN POWER SYSTEMS

Philip B. Smith
Ensign, United States Navy
BS, United States Naval Academy, 2021

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2022

Approved by: Wei Kang
Advisor

Thor Martinsen
Co-Advisor

Anthony J. Gannon
Second Reader

Francis X. Giraldo
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

How can we measure and classify neural network robustness across differently distributed data to avoid misuse of machine learning tools? This thesis adopts several metrics to measure the initial condition robustness of feedforward neural networks, allowing the creators of such networks to measure and refine their robustness and performance. This could allow highly robust neural networks to be used reliably on untrained data distributions and prevent the use of less robust networks as a black box in a poor environment. We test this measurement of robustness on a series of differently sized neural networks trained to detect and classify microgrid power system faults, giving examples of both robust and nonrobust networks, along with suggestions on how to maximize robustness. The analysis reveals that collecting data from segments along trajectories enhances the robustness of neural networks. In such data sets, the distribution of data points is dominated by the dynamics of the system, not the initial state distribution.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DOD	Department of Defense
ML	Machine Learning
NPS	Naval Postgraduate School
RMSE	Root Mean Squared Error
USN	U.S. Navy

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I owe special thanks to my advisors Professor Wei Kang and CDR Thor Martinsen for their endless support and help focusing my thoughts. This thesis would not have been possible without them.

Additionally, I would like to thank Visiting Professor Tron Omland of the Norwegian National Security Authority for his assistance reviewing my work. His contributions significantly increased the quality of my work.

Finally, I would like to thank my friends for their continuous advice and aid, and my family for their constant love.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Overview

Machine learning has become ubiquitous in many fields, from computer vision, to entertainment, to national defense. While we rely on these systems, not all of them are always robust enough for real-world cases. Adversarial attacks, data poisoning, or unpredicted levels or types of noise can lead to incorrect decisions, classifications, or predictions by the machine learning system without raising any red flags. In this thesis, we examine the effects of different types and levels of noise and uncertainties affecting both the initial conditions of systems and the data itself, examining the robustness of a traditional feed-forward neural network in an industrial setting.

Electrical microgrids were chosen as the field of study. As self-sufficient energy systems, microgrids can operate disconnected from larger interconnected energy grids, making them a popular choice for national defense, infrastructure, and other uses where reliability, redundancy, and efficiency is a priority. Since a microgrid is relatively small compared to traditional electric grids, its behavior is also easier to monitor and analyze.

Simulated models were mainly used, due to institutional constraints. These models simulated the operation of the rotors in a power generator running with three potential faults, or with no fault. Code for the simulations is included in Appendix A.1.

1.2 Motivations

This thesis attempts to both build feed-forward neural networks capable of monitoring and drawing conclusions about the behavior of microgrids and to test the robustness of these machine learning models against uncertainties of data distribution.

Attempting to monitor an electrical system's power consumption and metrics to draw conclusions about performance and system tasking is not new. The difference between a system being on or off should be visible to a human observer, but as a power system becomes

more complex the pattern recognition and classification required for task assignment is much more suited to machine learning. Machine learning tools able to identify tasks or performance of a component in a microgrid would be very useful.

Secondly, we are concerned about the robustness of machine learning in general, and our feed-forward neural network specifically. The desire for robustness includes the ability of a machine learning system to retain an acceptable degree of accuracy across different operating conditions. If an adversary influences the sampling, labelling, or probability distribution of even a small proportion of training or testing data, the performance of a model can be greatly affected. Robustness may also affect non-adversarial situations, such as the applicability of a model to a new environment or maintaining accuracy after an unanticipated shift in probability distributions. A robust model could be used as more of a "black box" in multiple situations and probability distributions, while non-robust models cannot be applied easily to different situations.

In this thesis, we will examine distributional robustness, which we define as *the quantifiable ability of a model to perform on a given distribution a measurable distance from the distribution on which the model was originally trained.* Distributional robustness can be viewed as a proportion of original performance on the trained dataset. To accomplish this, we will generate different data sets by sampling the initial condition variation of our power system from different distributions. While maximizing robustness may seem ideal, discovering that a model lacks robustness is also a very valuable finding, as it may prevent future classification errors.

CHAPTER 2:
Literature Review

Machine learning, or the applications of algorithms to draw inferences from data or solve problems without first principle models or explicit solutions, remains a widely popular and useful field of study. This thesis mainly focuses on feedforward neural networks, a basic yet useful machine learning tool, but the study of robustness can easily be extended to other machine learning tools. This chapter explains the development and function of feedforward neural networks, explores how distributional robustness applies, and examines the importance and use of electrical microgrids.

2.1 Feedforward Neural Networks

Designed to mirror the structure of a brain, feedforward neural networks are machine learning systems in which information moves from one set of neurons to another in a series of linear functions and activations. In a feedforward network, information only moves forward, without forming a cycle, thereby keeping the system much simpler than recurrent networks, which contain cycles. The network attempts to find $\hat{y}$, an approximation of ground truth y, given x as an input. To construct this neural network, we choose hyperparameters n_l layers, with s_l neurons in layer l, with n_l and s_l derived empirically. The first layer draws from the input x, and each neuron in all subsequent layers draw from the previous layer. Mathematically, our j^{th} neuron in layer $l+1$ outputs

$$f\Big(\sum_{i=1}^{s_l} W_{ji}^{(l)} a_i^{(l)} + b^{(l)}\Big) = f(z_j^{(l+1)}) = a_j^{(l+1)},$$

where s_l is the number of neurons in layer l, $a^{(l)}$ is the set of outputs from layer l, and $W_{ij}^{(l)}$ represents the weight of the connection between neuron i in layer l and neuron j in layer $l+1$. $b_i^{(l)}$ is the bias layer connected to unit i in layer $l+1$. The function f is our activation function, which is usually the sigmoid, hyperbolic tangent, or Rectified Linear Unit (ReLU) function [1]. A neural network is constructed by linking collections of nodes in layers, visible in Figure 2.1.

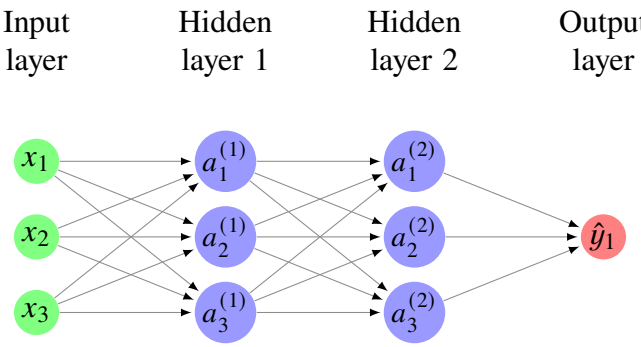


Figure 2.1. *An example neural network with two hidden layers and three nodes per hidden layer.*

To find our matrix W of weights, we use a process called backpropagation [2], based on gradient descent to reduce a cost function. For a training set of m examples, we define our cost function as

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||_2^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2,$$

where $h_{W,b}(x^{(i)})$ is the final output of our neural network trained on a fixed training set $\{(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})\}$ and λ is a regularization term for the weights, preventing overfitting.

To determine the W and b which will minimize J, we first initialize each parameter $W_{ij}^{(l)}$ and each $b_i^{(l)}$ as a random value near zero to accomplish symmetry breaking. While there are many ways to train neural networks, we chose to use the following gradient descent method to illustrate the training process. We update the parameters W and b as

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

and

$$b_i^{(l)} := b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b).$$

The partial derivatives are computed as follows. Letting $a^{(1)} = x$, with x as our input data, we compute our generalized forward propagation as

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

and

$$a^{(l+1)} = f(z^{(l+1)}).$$

At the output layer of our network, we set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \circ f'(z^{(n_l)}),$$

where $\circ$ represents the Hadamard product of two matrices. For $l = n_l - 1, n_l - 2, ..., 2$, we set

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \circ f'(z^{(l)}).$$

We can then compute the partial derivatives

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)}(a^{(l)})^T$$

and

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

These equations come together to give us a neural network like the one pictured in Figure 2.1.

2.2 Applications of Machine Learning

In recent years, machine learning has become one of the fastest growing fields. The ability to use algorithms and statistical models to analyze, learn from, and draw conclusions from large quantities of data has proved useful across many different fields. Cybersecurity applications

include improvements to network routing decisions, identification of suspicious activity, and protection of AI computer vision from data poisoning.

Machine learning systems have proven to be very useful in detecting irregular activities on a network leading up to an attack. Lu, Mabu, Wang, and Hirasawa developed a class association rule pruning system based on matching degree and genetic algorithms to unify misuse detection—detecting the beginning of known attacks on a computer system—and anomaly detection–detecting suspicious activity not connected to known attacks [3]. This helped reduce the flaws of misuse detection and anomaly detection, which are weaknesses in detecting new attacks and a high false positive rate, respectively.

Symmetric key cryptologic systems have also benefited from the application of machine learning. Yu and Cao proposed using time varying delayed Hopfield neural networks to form sufficiently pseudorandom binary strings used to encrypt large multi-media files [4]. Once the neural networks finish, this chaotic communication encryption has a much simpler encryption and decryption process compared to public key encryption.

Machine learning has been extraordinarily useful in computer vision. Caelli and Bischof provide an excellent in-depth overview of this complex field, discussing how to use complex algorithms to encode images, extract important features, and accurately associate these features with specific human knowledge. Machine learning was especially helpful for the rule generation step, in which the system makes steps to generate descriptions of objects [5]. While both computing power and new machine learning systems have developed in the years since Caelli and Bischof published this book, it still forms a very solid basis for understanding the applications of machine learning to computer vision.

Additionally, machine learning can be used to protect machine learning from data poisoning. Data poisoning occurs when an adversary tries to manipulate a machine learning system by changing points in the training data set so the model will mislabel certain instances. Barreno, Nelson, Joseph and Tygar explored several protections against such attacks, including training a model to use the Reject On Negative Impact (RINO) defense, in which training examples are discarded if they have a negative impact on performance [6]. There are still several issues with defense, such as an increase in computational cost and a decrease in accuracy, but machine learning systems can optimize these costs.

Machine learning has been implemented in protecting industry and infrastructure from cyberattacks in recent years. Linda, Vollmer, and Manic tailored an intrusion detection system for specific applications in critical infrastructures. Using neural networks for cluster boundary modeling, their IDS-NNM model managed to capture all intrusion attempts using previously unseen data rather efficiently [7]. They envisioned applications in nuclear plants, power systems, or other supervisory control and data acquisition (SCADA) fields, especially since neural network-based models may be able to detect new attacks that a trained person would not.

The incorporation of these models, however, can pose an additional threat. Recognizing that these machine learning systems can provide an additional vector of attack into the system, Anthi, Williams, Rhode, Burnap, and Wedgbury explored the different ways in which machine learning could be used to attack machine learning based intrusion detection systems protecting industrial control systems such as power grids and manufacturing plants. Their adversarial machine learning attacks decreased the performance of the systems protecting simulated critical infrastructure by about 10%, but this loss was partially reversed by training the detection systems to protect against adversarial attacks [8].

2.2.1 Distributional Robustness

Due to a variety of reasons, from operator error to adversarial influence or simply being the best available option, neural networks may sometimes be used outside of the environment in which they were trained. This will negatively impact their performance to an uncertain degree. Quantifying this degree—measuring the robustness of the network—is essential.

More broadly, the desire for robustness includes the ability of a machine learning system to remain stable across and outside of different probability distributions. Non-robust models will react poorly to adversarial influence or probability shifts, with large performance changes resulting from small shifts to the sampling, labelling, or probability distribution of small proportions of training or testing data. In this paper, we will define distributional robustness as *the quantifiable ability of a model to perform on a given distribution a measurable distance from the distribution on which the model was originally trained.* Distributional robustness can be viewed as a proportion of original performance on the trained dataset.

Many attempts at creating robust learners and optimizers built on unknown data distributions rely either on known moments (such as mean and covariance) or on making worst-case estimates within a limited bound of possible data distributions. Gao and Kleywegt came up with a statistical-distance-based Distributionally Robust Stochastic Optimizer (DRSO) that allows consideration of otherwise unwieldy data distributions [9]. By considering the Wasserstein metric described in Definition 1 [10], Gao and Kleywegt were able to obtain precise structural descriptions of worst-case distributions, allowing model creators to hedge bets against likely and unlikely statistical distributions.

Definition 1 *Let (Ξ, d) be a Polish space. For $p \geq 1$, let $P_p(\Xi)$ denote the collection of all probability measures μ on Ξ with a finite p^{th} moment. Then, there exists some x_0 in Ξ such that*

$$\int_{\Xi} d(x, x_0)^p \, d\mu(x) < \infty.$$

The p^{th} Wasserstein distance between two probability measures μ and ν in $P_p(\Xi)$ is defined as

$$W_p(\mu, \nu) := \Big(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\Xi \times \Xi} d(x, y)^p \, d\gamma(x, y) \Big)^{\frac{1}{p}},$$

where $\Gamma(\mu, \nu)$ represents a collection of all measures on $\Xi \times \Xi$ with marginals μ and ν on the first and second factors, respectively. Here, a Polish space is a space homeomorphic to a complete metric space with a countable dense subset. This definition, however, is too complex for this paper, as we will only be concerned with the second Wasserstein distance between two one-dimensional probability distributions. Since all probability distributions integrate to 1, the 2nd Wasserstein distance between probability measures μ_1 and μ_2 simplifies to

$$W_2(\mu_1, \mu_2) = \left(\int |F_1^{-1}(x) - F_2^{-1}(x)|^2 \, dx \right)^{1/2}, \tag{2.1}$$

where F_1^{-1} and F_2^{-1} are the inverse cumulative distribution functions, or quantile functions, of μ_1 and μ_2, respectively.

Other choices in model development may increase robustness with regards to distribution. While self-supervised models may be slightly less efficient and accurate than fully supervised models, Hendrycks, Mazeika, Kadavath, and Song demonstrated that self-supervision

increases robustness to both adversarial examples and dealing with near-distribution and out-of-distribution data points [11]. When used in conjunction with a fully supervised system, the self-supervised system had minimal impact on accuracy while improving robust performance without requiring larger models or additional data.

Importantly, higher robustness is not always necessary or desired in machine learning systems. Several papers found that an increase in adversarial robustness in the ML tool, defined as resistance against adversarial attacks against the training or classification phase, can also come with a decrease to performance on the original data set, since the model is trained to prioritize more than simple performance [12]. Different machine learning systems may require different levels of robustness—it is more important to have a level of robustness appropriate for a specific scenario. It is unclear if this drop in performance will still present itself when studying distributional robustness.

2.3 Applications to Microgrids

As countries and organizations try to modernize power grids, microgrids have shown great potential. A microgrid is a small electrical network containing both power generators and power consumers that is usually attached to a broader grid but able to function independently if required. While microgrids are extraordinarily useful for small to medium scale electrical system management, the changing nature of a microgrid under load leaves it particularly vulnerable to electrical faults [13]. Machine learning seems like an appropriate solution for fault detection, since it has the ability to quickly process and act on a large amount of data.

Traditional microgrid fault detection has not proven to be noise-immune or precise enough for some microgrid applications. Already, efficient machine learning solutions have been proposed. One of the newest and most promising solutions involves using multiple layers of restricted Boltzmann machines to construct a new discrete-wavelet transform system [14]. While this is overly complex for inclusion in this thesis, it is worth noting that each restricted Boltzmann machine is trained with an artificial neural network, and is significantly more noise tolerant than currently used commercial models.

While simpler models are not always as noise tolerant, they can still be effective for fault detection and mitigation. Neural networks by themselves have been shown to efficiently

detect faults in both data generated from models and real-world microgrids [13]. In fact, some simple neural networks can detect faults fast enough to isolate affected areas before damage is done to the rest of the microgrid [15]. It is therefore important to examine the distributional robustness of such simple neural networks.

CHAPTER 3:
Methodology

3.1 Data Generation and Structure

Our data was simulated as a traditional IEEE 9-bus model with three power generators, numbered 1, 2, and 3 [16]. An example is shown in Figure 3.1.

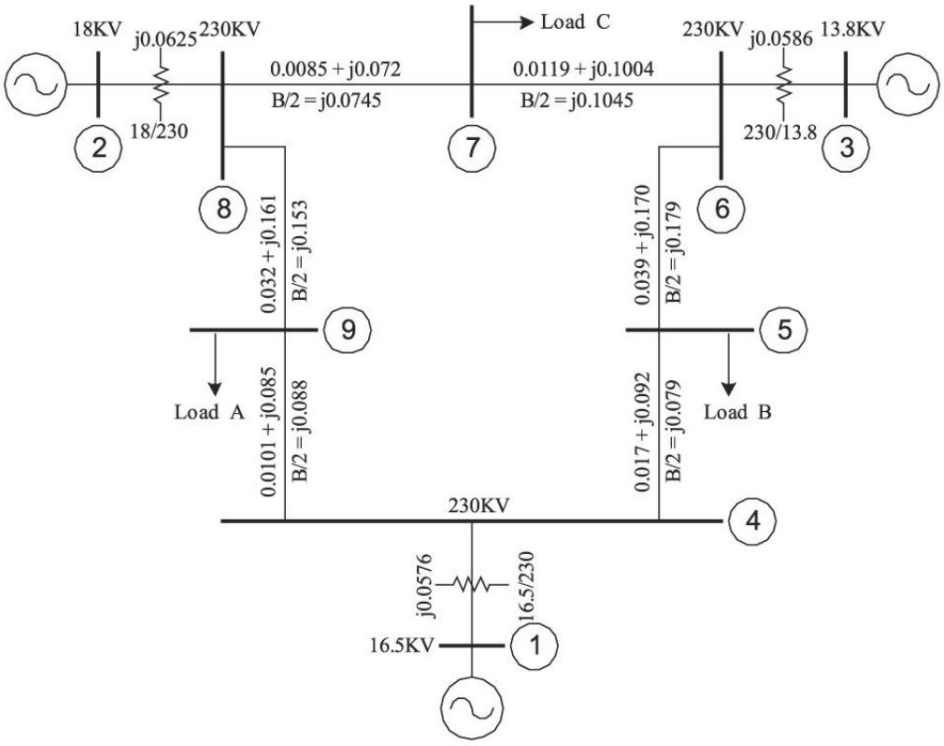


Figure 3.1. *A 9-bus power system model. Source: [17].*

Three fault types in this power system are possible. A fault between generators one and two is designated Fault 1, between generators two and three is designated Fault 2, and between generators three and one is designated Fault 3. The goal is to identify a fault using deep learning with one second of data of the trajectories of angular velocities.

Rotor angle speed in radians per second, the rotor angle in radians, and time related according to the following differential equations.

$$\frac{2H_i}{\omega_R}\frac{d\omega_i}{dt} + D_i\omega_i = P_{mi} - P_{ei}, \quad i = 1, 2, 3$$

$$\frac{d\delta_i}{dt} = \omega_i - \omega_R$$

$$P_{ei} = E_i^2 G_{ii} + \sum_{j=1, j\neq i}^{n} E_i E_j Y_{ij} cos(\theta_{ij} - \delta_i + \delta_j),$$

Here, H_i is the generators' stored kinetic energy, ω_R is the angular velocity, ω_i is the angular velocity of the generator, D_i is a drag coefficient, and δ_i is the generators rotor angular position. The two power components, P_{mi} and P_{ei}, are the mechanical and electrical energy, respectively, for generator i. E_i is the generator's constant excitation voltage. G_{ii} is the conductance of each generator at term i [18]. The rotor angle equilibrium point for generators 1, 2, and 3 are 0.0396, 0.3444, and 0.2300 radians, respectively. The angular velocity of the system is 120π. MATLAB data generation code is included in Appendix A.1.

A series of five-second trajectories of each generator measured at 10 Hz were generated in MATLAB, with the initial rotor angle of each generator chosen from a uniform distribution between $[-0.05, 0.05]$ around the equilibrium for each generator. Each trajectory had an equal chance of experiencing Fault 1, 2, 3, or no fault at time $t = 0$, following a uniform distribution.

From each trajectory, a one-second time window was chosen following a uniform distribution, giving 10 data measurements from each generator. These were concatenated, so our

final data structure for n trajectories is

$$\begin{bmatrix} \omega_1^{(1)}(t_1) & \omega_1^{(2)}(t_1) & \dots & \omega_1^{(n)}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ \omega_1^{(1)}(t_{10}) & \omega_1^{(2)}(t_{10}) & \dots & \omega_1^{(n)}(t_{10}) \\ \omega_2^{(1)}(t_1) & \omega_2^{(2)}(t_1) & \dots & \omega_2^{(n)}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ \omega_2^{(1)}(t_{10}) & \omega_2^{(2)}(t_{10}) & \dots & \omega_2^{(n)}(t_{10}) \\ \omega_3^{(1)}(t_1) & \omega_3^{(2)}(t_1) & \dots & \omega_3^{(n)}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ \omega_3^{(1)}(t_{10}) & \omega_3^{(2)}(t_{10}) & \dots & \omega_3^{(n)}(t_{10}) \end{bmatrix},$$

where $\omega_a^{(b)}(t_c)$ represents the angular velocity of rotor a during trial b at time t_c. After normalization, this matrix is our input matrix to our neural network.

An output matrix of each trajectory's fault classification was labeled as integers. Each fault was represented by the number associated with it, and no fault was represented by the number 0. This matrix was

$$\begin{bmatrix} k_1 & k_2 & \dots & k_n \end{bmatrix}.$$

3.2 Lower Dimensional Distances

To test robustness, we began by attempting to find the furthest initial conditions from the data on which our network was trained. Our simulated power system data uses a uniform distribution in its initial condition variation. Our data, however, is highly dimensional, and the Wasserstein distance defined in Definition 1 is not always computationally efficient in high dimensions.

Instead, we illustrate the distance between distributions in one-dimensional space. Since our simulated data uses a uniform distribution, we created a one-dimensional uniform distribution against which we tested many other statistical distributions, with several constraints, including data concatenation at the bounds of the uniform distribution, calculating distances numerically [19]. Within these constraints, we searched for distributions that were not close

to a uniform distribution on $[-0.05, 0.05]$ when using the 2^{nd} Wasserstein distance defined in Definition 1 and Equation 2.1. Some probability distributions are visible in Figure 3.2.

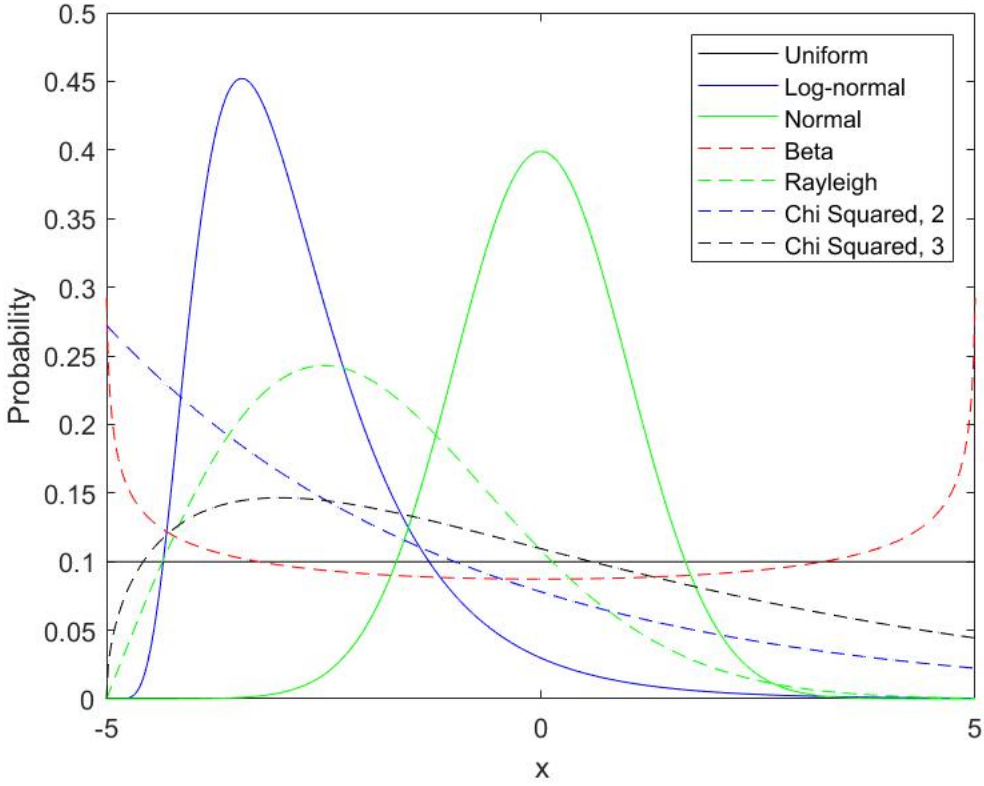


Figure 3.2. *The probability density functions of various considered initial conditions, scaled onto the domain $x = [-5, 5]$ for clarity. Significant distributions are solid lines, while the rejected distributions are dotted lines.*

Our first chosen distribution was a normal distribution with $\mu = 0$ and $\sigma = 0.01$, chosen to minimize the amount of data concatenated at the edges. This distribution had a 2^{nd} Wasserstein distance of 13.49 from the uniform distribution on $[-0.05, 0.05]$. Its variance is 0.1001. Our second chosen distribution was a log-normal distribution, created from a normal distribution with $\mu = 0$ and $\sigma = 0.1$, then scaled onto $[-0.05, 0.05]$. This distribution was slightly further from the uniform distribution than the normal distribution was, with a 2^{nd} Wasserstein distance of 18.01. Its variance was 0.1018. The covariance between these two variables is -0.0002.

3.3 Simulated Data

3.3.1 Network Construction

We first generated data to construct and optimize an initial neural network. We began with $5,000$ trajectories each for Faults $1, 2, 3$, and no fault. Our data points were drawn from the trajectories as described in Section 3.1. We studied a random one-second section of each trajectory, taking measurements from all three generators at 10 Hz, giving us a $30 \times 20,000$ input data set on which to train. Our predictions would attempt to match a $1 \times 20,000$ vector of fault numbers. To find the optimal network size, we trained and tested neural networks between one and five layers, between 1 and 16 nodes per layer, using the hyperbolic tangent, sigmoid, and rectified linear activation functions. In this paper, we will focus on the hyperbolic tangent function, the highest performing activation function. Neural networks were constructed using the MATLAB nntraintool, which allows users to create a custom neural network with full control over hyperparameters, training method, and activation function. Optimization was done with the Levenberg–Marquardt algorithm. Our simplest optimum network was found with four hidden layers of eight nodes per layer using the hyperbolic tangent function, which achieved perfect predictions over both the training data set and a similarly sized, independently developed initial validation set. Code for training all neural networks is included in Appendix A.2.

Next, we began to test our neural network on differently distributed data. Our original initial condition was a uniform distribution of initial rotor angles between -0.05 and 0.05 radians around the equilibrium. Based on the distances calculated in Section 3.2, we generated more testing data sets using normal and log-normal distributions designed to be a large distance from our training distribution. All distributions were then truncated to fit within $[-0.05, 0.05]$. Once again, our normal distribution used $\mu = 0$ and $\sigma = 0.01$ and the log-normal distribution used $\mu = 0$ and $\sigma = 0.1$, linearly transformed into the domain of $[-0.05, 0.05]$. Generating a 30×20000 testing set for each of these, our initial neural network performed perfectly again.

Next, we examined network performance on differently distributed testing sets. Our uniform distribution we trained our network on had a mean of 0 and a standard deviation of $\sqrt{1/1200}$. Trying to match averages and standard deviations of our normal and log-normal distributions to our training uniform distributions, our normal distribution used $\mu = 0$ and $\sigma = \sqrt{1/1200}$,

and the log-normal distribution used $\mu = \log(0.05) - \log(4/3)/2$ and $\sigma = \sqrt{\log(4/3)}$, linearly transformed into the domain of $[-0.05, 0.05]$. Again, perfect performance was achieved.

Finally, we varied the size of the network, studying both different depths and widths. Near perfect performance was achieved here, with the only errors coming from massive changes to our network. Results for this are included in Section 4.3.

While perfect performance on these sets may seem surprising at first, we thought this may have been due to our initial choices of initial condition distributions and network hyperparameters. Since we sample from a random one second interval in the trajectory instead of directly at the initial condition, it is possible that our interval-based sampling was the cause of the robustness.

3.3.2 Dynamics Analysis

How could we tell if the dynamic nature of our system was dominating the initial conditions of our power system? To answer this question, we decided to analyze the initial conditions of the one second portion of the trajectory we actually used in our neural network. To do this, we compared histograms of the rotor angle of all three generators both at $t = 0$ and at the beginning of a random one-second interval for uniform, normal, and log-normal distributions.

Our theory was that the dynamic nature of our problem would force most trajectories to behave similarly. If this was true, the rotor angle at $t = 0$ would follow the given distribution around the equilibrium point of the rotor, but the beginning of a randomly selected window for a given generator would look the same, regardless of initial condition distribution. If the histograms were significantly different, however, this might suggest that the dynamics of our microgrid do not dominate our initial condition variation, and that something else may be the cause of our high level of distributional robustness.

CHAPTER 4:
Results

4.1 Network Construction

Our first task was to optimize our neural network's parameters and hyperparameters to achieve the best possible predictions. Our two performance metrics are root mean squared error (RMSE), (the square root of the sum of the square of the difference between predictions and ground truth divided by the number of samples) and misclassification rate, which is the proportion of incorrectly classified trajectories. A misclassification rate of 0 means every trajectory was correctly classified. For reference, the RMSE of random predictions is just over 1.58. Iterating over the number of hidden layers and nodes per layer, we found the RMSE and misclassification rates visible in Tables 4.1 and 4.2, respectively.

		Nodes per Hidden Layer					
		2	4	6	8	10	12
Hidden Layers	1	0.2202	0.0173	0.0679	0.1684	0.1001	0.1815
	2	0.2643	0.0674	0.1489	0.2385	0.1142	0.0916
	3	0.1797	0.2604	0.1471	0.1327	0.2291	0.0368
	4	0.0381	0.0000	0.0227	**0.0137**	0.1421	0.0322
	5	0.2226	0.0003	0.1946	0.1070	0.0009	0.0227

Table 4.1. Root mean squared errors of different sized neural networks on a validation data set with uniformly distributed initial conditions, with the chosen network bolded.

We can see an upper triangular pattern in Table 4.2, with higher performing networks clustered in the upper right of the tables. This structure suggests any increase in hidden layers should also be accompanied by an increase in nodes per layer to avoid an increase in misclassification. To take full advantage of the upper triangular structure in both tables, we chose to use a network with four hidden layers of eight nodes each as our base best neural network, which consistently performed the best. This network is visualized in Figure 4.1, and performed very effectively on all data distributions. On the validation data set with uniformly distributed initial conditions, the RMSE was 3.72×10^{-2}, and the misclassification

		Nodes per Hidden Layer					
		2	4	6	8	10	12
Hidden Layers	1	0.04145	0	0	0	0	0
	2	0.0982	0	0	0	0	0
	3	0.0364	0.10665	0	0	0	0
	4	0	0	0.00065	**0**	0	0
	5	0.04835	0	0.0432	0.01225	0	0

Table 4.2. Misclassification rate of differently sized neural networks on a validation data set with uniformly distributed initial conditions, with the chosen neural network bolded. Note the upper triangular structure in both this table and Table 4.1.

rate was 0. The RMSE was 1.27×10^{-2} and the misclassification rate was 0 on validation data with normally distributed initial conditions. On the validation data set with log-normally distributed initial conditions, the RMSE was 1.26×10^{-2}, and the misclassification rate was 0. A histogram of error predictions for all three distributions is included in Figure 4.2. Examining these histograms, we see a pattern in all predictions across distributions. Our model predicts no fault and Fault 1 cases with almost perfect accuracy across uniform, normal, and log-normal distributions, but the error increases when it considers Faults 2 and 3.

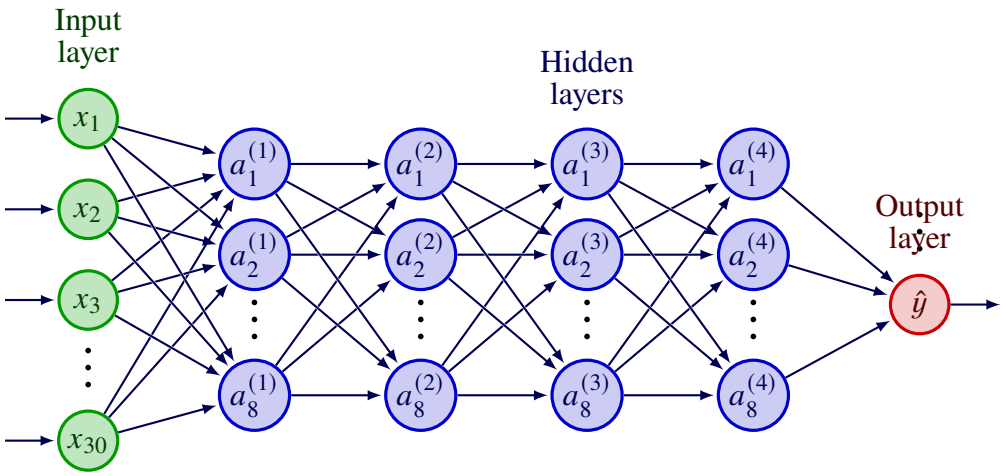
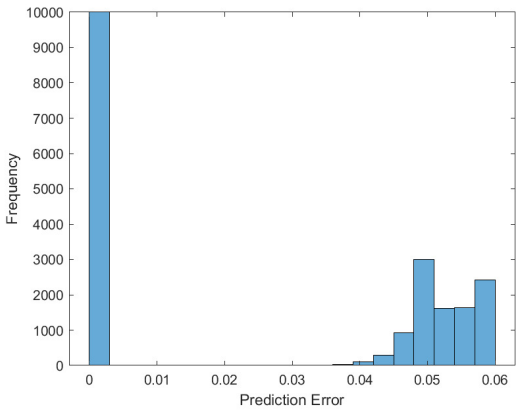
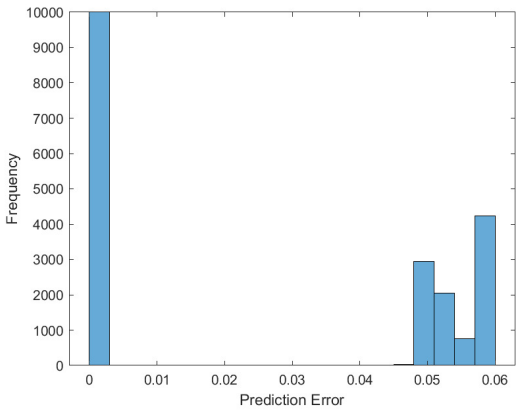


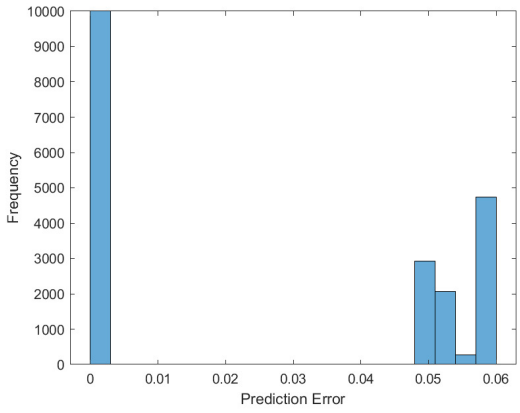
Figure 4.1. *A visualization of our first optimum neural network, with four hidden layers and eight nodes per layer. Bias terms have been omitted for clarity.*



(a) *Uniform distribution*



(b) *Normal distribution,* $\mu = 0$, $\sigma = 0.1$



(c) *Log-normal distribution,* $\mu = 0$, $\sigma = 0.01$, *scaled*

Figure 4.2. *Error histograms of our base network visualized in Figure 4.1 on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

We were rather surprised to see such high performance across distributions, but on closer examination it seems to make sense. Our network is able to perfectly predict most trajectories that have an initial condition variation of $[-0.05, 0.05]$ around the equilibrium, and both our normal and our log-normal distributions exclusively produce trajectories that begin in that range. Another possible explanation for the network's accuracy across initial condition distributions comes from the sampling method and the dynamic nature of our problem. Since each input comes from a random one second window from a five second trajectory, it

is possible the initial condition variation is dominated by the dynamics of the power system, causing most trajectories to behave similarly to each other regardless of initial condition variation.

We decided to change both our distributions and our network hyperparameters to try and explain this accuracy. First, we will examine if our base network with four hidden layers of eight nodes each loses performance as we apply it to distributions further and further away from the uniform distribution it was trained on. Next, we will apply differently sized networks to the three distributions our base network performed so well on, to see if our network just happened to be robust, or if other networks perform similarly.

4.2 Distributional Changes

We tested the neural network visualized in Figure 4.1 against data with different normally and log-normally distributed initial conditions. If our network performed similarly on additional distributions that are not close to the trained uniform distribution, this would suggest our network was robust. Again, if an initial condition from a distribution was chosen above or below of the allowed range of $[-0.05, 0.05]$, it was assigned the value 0.05 or -0.05, depending on which side of the range it fell.

Our first changed distribution was a normal distribution modeled off of our uniform distribution. This normal distribution had the same mean and standard deviation as the original training uniform distribution, with $\mu = 0$ and $\sigma = \sqrt{1/1200} \approxeq 0.0289$. Using our original network composed of four hidden layers of eight nodes each, our RMSE was 0.119 and our misclassification rate was 0.

Next, we tried different log-normal distributions. We again matched the mean and standard deviation to the training uniform distribution instead of choosing distribution parameters empirically by distance. Since a log-normal distribution is skewed to the right, however, we created two log-normal distributions, which were mirror images of each other, each with $\mu = 0$ and $\sigma = \sqrt{1/1200} \approxeq 0.0289$. To create a log-normal distribution X, we considered a normal distribution $Y = log(X)$, with $\mu_Y = \log(0.05) - \log(4/3)/2 \approxeq -3.140$ and $\sigma_Y = \sqrt{\log(4/3)} \approxeq 0.536$. This created a single, right skew log-normal distribution X with a standard deviation that matches the original uniform distribution, but with $\mu = 0.05$.

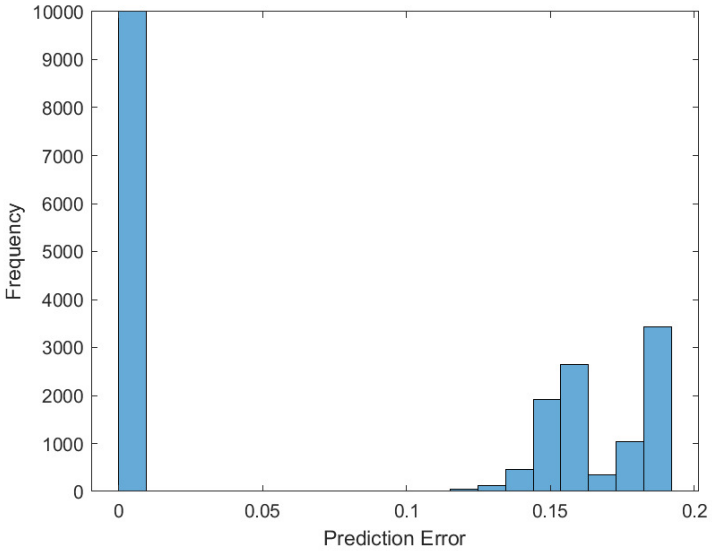
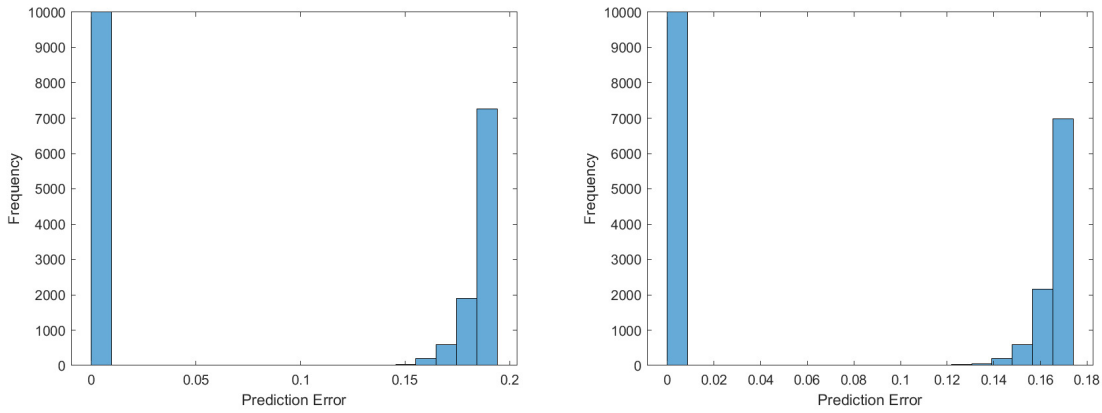


Figure 4.3. *Prediction errors of a neural network composed of four hidden layers of eight nodes each on normally distributed data with* $\mu = 0$ *and* $\sigma = \sqrt{1/1200}$.

To compensate for the skew and center distributions around 0, we considered the two distributions $X_1 = X - 0.05$ and $X_2 = -X + 0.05$, producing a right-skewed and a left-skewed log-normal distribution, respectively, with the correct mean and standard deviation.

We ran each distribution through our base neural network composed of four hidden layers of eight nodes each. On the right-skew X_1 distribution, our RMSE was 0.117, and on our left-skew X_2 distribution, our RMSE was 0.131. The network had a misclassification rate of 0 on both distributions. Histograms of error predictions are included in Figure 4.4.

To summarize, widening the standard deviation made our predictions slightly less accurate, but a similar error pattern is still clearly visible in the histogram in Figure 4.3. While the error groupings were somewhat condensed in Figure 4.4, the pattern of a section of near-perfect predictions followed by a group of much less accurate predictions continues. Put together, the increase in prediction error does suggest our network is slightly less robust than we earlier thought, but a pattern is still visible.



(a) *Left-skew lognormal distribution* **(b)** *Right-skew lognormal distribution*

Figure 4.4. *Error histograms of network performance on data with left- and right-skew log-normal initial condition variation, with distribution parameters matching training. This is accomplished by using parameters* $\mu_Y = \log(0.05) - \log(4/3)/2$ *and* $\sigma_Y = \sqrt{\log(4/3)}$*, then shifting to match the domain.*

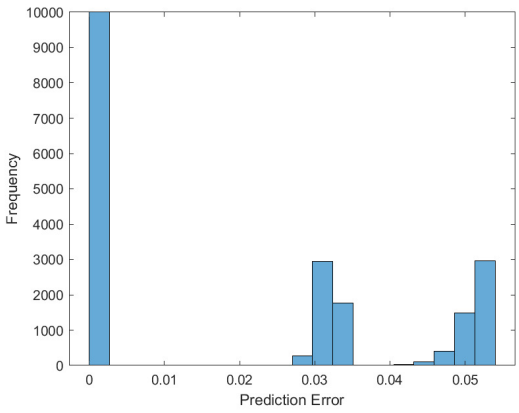
4.3 Depth and Width Changes

Our next consideration was to examine the influence of the depth or width of the neural network on distributional robustness. A network's depth is equal to the total number of layers in the network, except for the input and output layers, and the width of a layer is the number of nodes in that layer. Since we have the same number of nodes per layer in all our networks, we can assign a specific width to each network. In this section, we will use the normal and log-normal distributions discussed in Section 4.1.

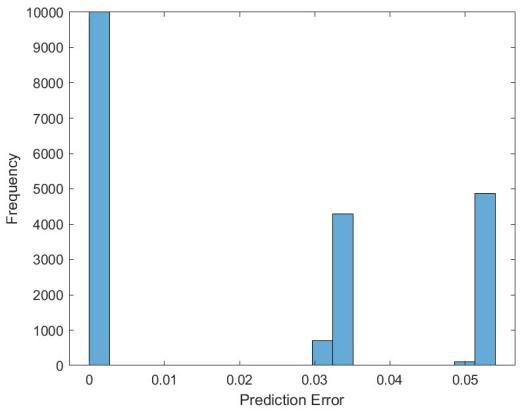
4.3.1 Depth Changes

We first attempted to vary the depth of our network. Shallower networks are less complex, and therefore not always as capable as their deeper counterparts, but are usually faster to train. We began by cutting our network from a depth of 4 hidden layers to 3, with the following results.

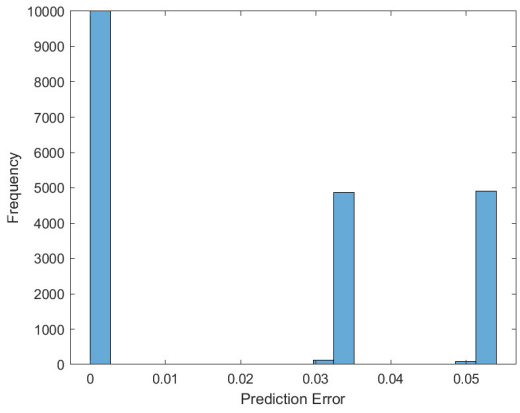
On the validation data set with uniformly distributed initial conditions, the RMSE was 3.01×10^{-2}, and the misclassification rate was 0. The RMSE was 3.09×10^{-2}, and the misclassification rate was 0 on validation data with normally distributed initial conditions.



(a) *Uniform distribution*



(b) *Normal distribution*

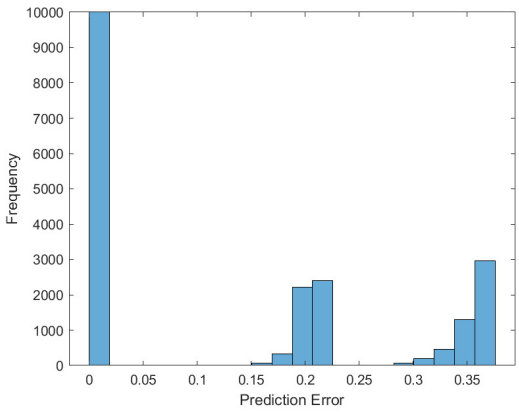


(c) *Log-normal distribution*

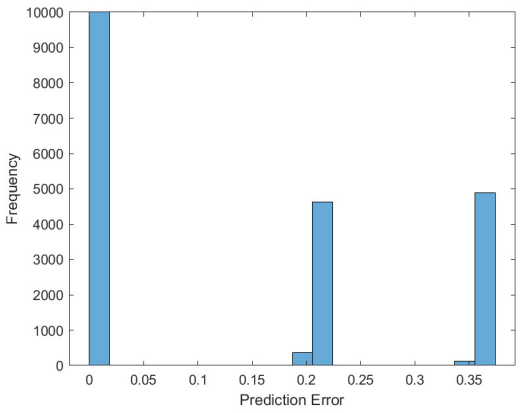
Figure 4.5. *Error histograms of a network with three hidden layers of eight nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

On the validation data set with log-normally distributed initial conditions, the RMSE was 3.10×10^{-2}, and the misclassification rate was 0. A histogram of error predictions for all distributions is included in Figure 4.5.

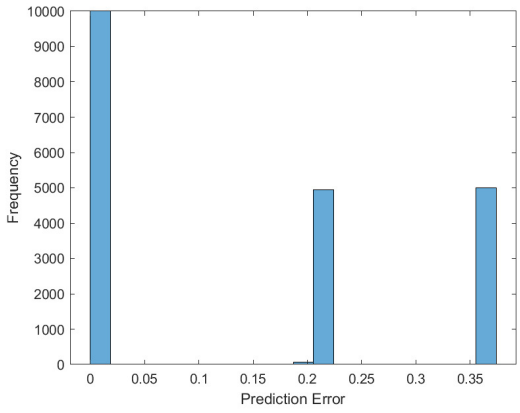
This adjustment to depth, however, did not have enough of an effect on network robustness. We next tried an even shallower network, using two hidden layers of eight nodes each. On uniformly distributed initial conditioned data, this network had a RMSE of 0.205, and a misclassification rate of 0. On normally distributed initial conditioned data, the network



(a) *Uniform distribution*



(b) *Normal distribution*

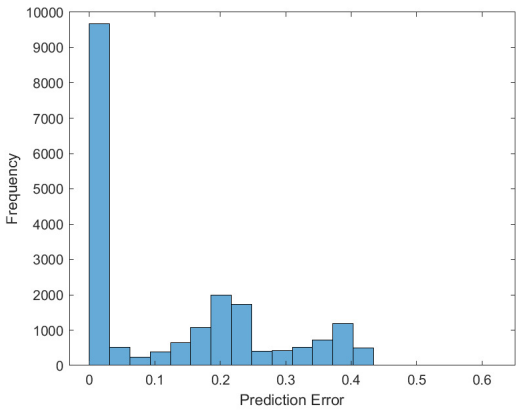


(c) *Log-normal distribution*

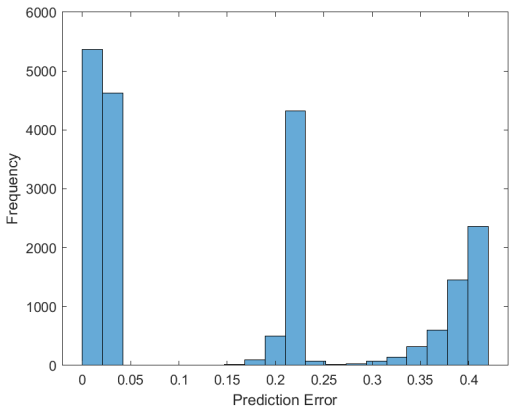
Figure 4.6. *Error histograms of a network with two hidden layers of eight nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

had a RMSE of 0.211 and a misclassification rate of 0. And on log-normally distributed initial condition data, our network had a RMSE of 0.212 and a misclassification rate of 0. A histogram of prediction errors is included in Figure 4.6.

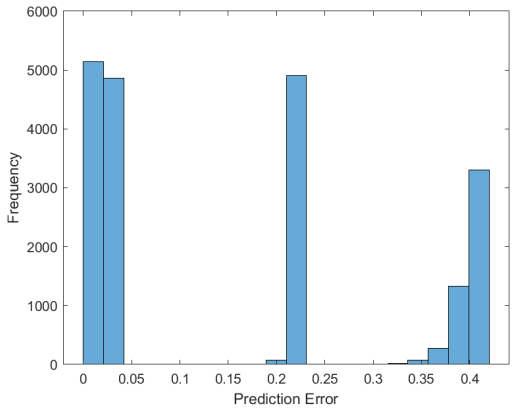
We finally attempted to see the performance of a network consisting of only a single layer of eight neurons. On data with a uniformly distributed initial condition, the RMSE was 0.187 and the misclassification rate was 5×10^{-5}. On data with normally distributed initial condition, the RMSE was 0.224 and the misclassification rate was 0. And on data with



(a) *Uniform distribution*



(b) *Normal distribution*



(c) *Log-normal distribution*

Figure 4.7. *Error histograms of a network with a single hidden layer of eight nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

log-normally distributed initial condition, our RMSE was 0.230 and the misclassification rate was 0. Error prediction histograms are included in Figure 4.7.

As we cut the depth of networks, our misclassification rate remained almost perfect, which means our predictions remained the same, but we did still experience a slight loss in robustness. In our histograms, we are able to see that the predictions of shallower networks tended to be less accurate, even if the rounded predictions were still perfect. Still, we wanted to find a change to the network that would more broadly affect robustness.

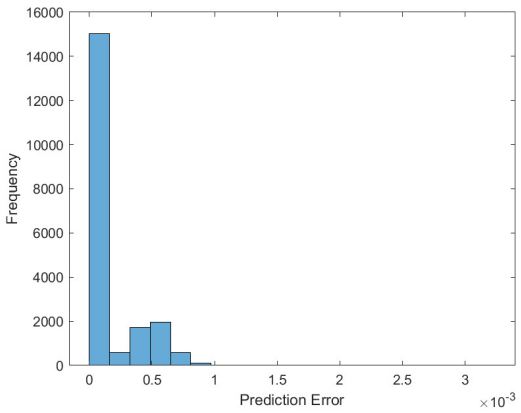
Another interesting observation comes from the shape of the histograms. Similar to the base network of four hidden layers of eight nodes, all networks predict no fault and Fault 1 cases with a very high degree of accuracy, but Fault 2 and 3 predictions are less accurate, even if they still round to the correct number. This produces the pattern visible in all histograms, regardless of distribution or network structure, of one tall section of good predictions, followed by two subsequent sections of deteriorating predictions. This suggests that the error pattern is consistent across distribution and network hyperparameters, only increasing as the validation distribution increases distance from the trained distribution and as the network hyperparameters move away from the optimal choice.

4.3.2 Width Changes

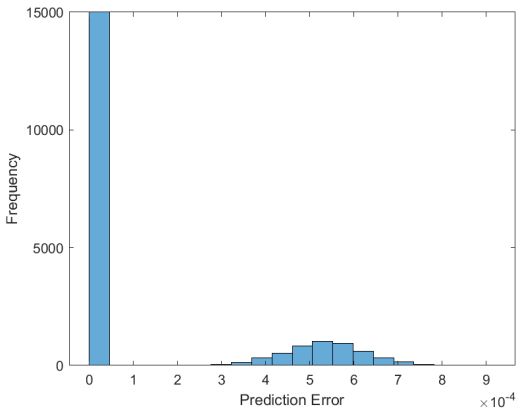
To verify these observations, we next varied the width of our network. We began by cutting the width to four nodes, still using four hidden layers and again tested on uniform, normal, and log-normal validation sets. On uniformly distributed initial conditioned data, our RMSE was 2.62×10^{-4} and our misclassification rate was 0. On data with a normally distributed initial condition, our RMSE was 2.71×10^{-4} and our misclassification rate was 0. Finally, on data with a log-normally distributed initial condition, our RMSE was 2.68×10^{-4} with a misclassification rate of 0. Error histograms are included in Figure 4.8. While we were somewhat surprised to see that this actually outperforms our original base network of four layers of eight nodes, this trend should not continue when looking at narrower neural networks.

We confirmed this by testing a network composed of four layers of two nodes each. On uniformly distributed initial conditioned data, this network's RMSE was 0.117 and the misclassification rate was 0. On data with a normally distributed initial condition, our RMSE was 0.120 and our misclassification rate was 0. Whereas on data with a log-normally distributed initial condition, our RMSE was 0.120 with a misclassification rate of 0. Error histograms are included in Figure 4.9.

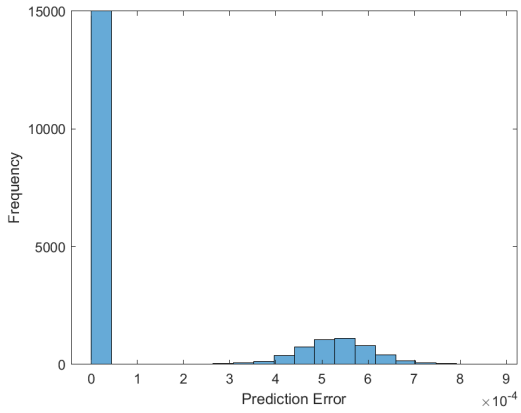
We were somewhat surprised to see the increased accuracy in the predictions from the network of four layers and four nodes, visible in Figure 4.8. However, the conclusions proposed when talking about network depth still tentatively hold when the width is changed,



(a) *Uniform distribution*



(b) *Normal distribution*



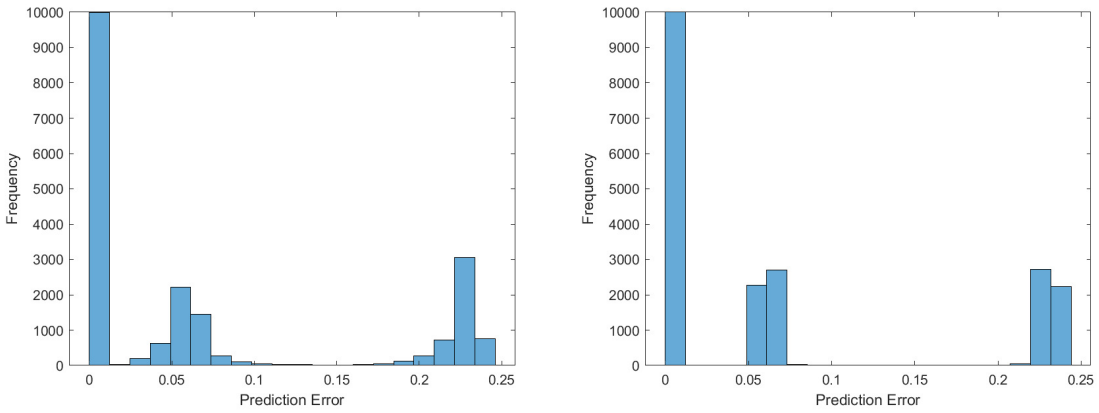
(c) *Log-normal distribution*

Figure 4.8. *Error histograms of a network with four hidden layers of four nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

and the normal pattern in the histograms reestablishes itself in Figure 4.9 as the network gets narrower.

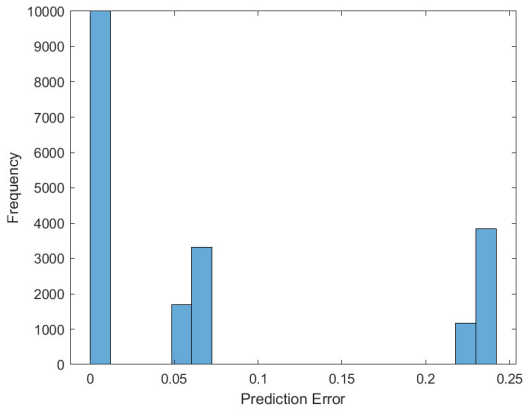
4.3.3 Depth and Width Changes

While we will not analyze multiple examples, it is worth considering what happens when we vary both the depth and the width of a network at once, to see if these effects would compound. To check this, we trained a network composed of three layers of four nodes each,



(a) *Uniform distribution*

(b) *Normal distribution*

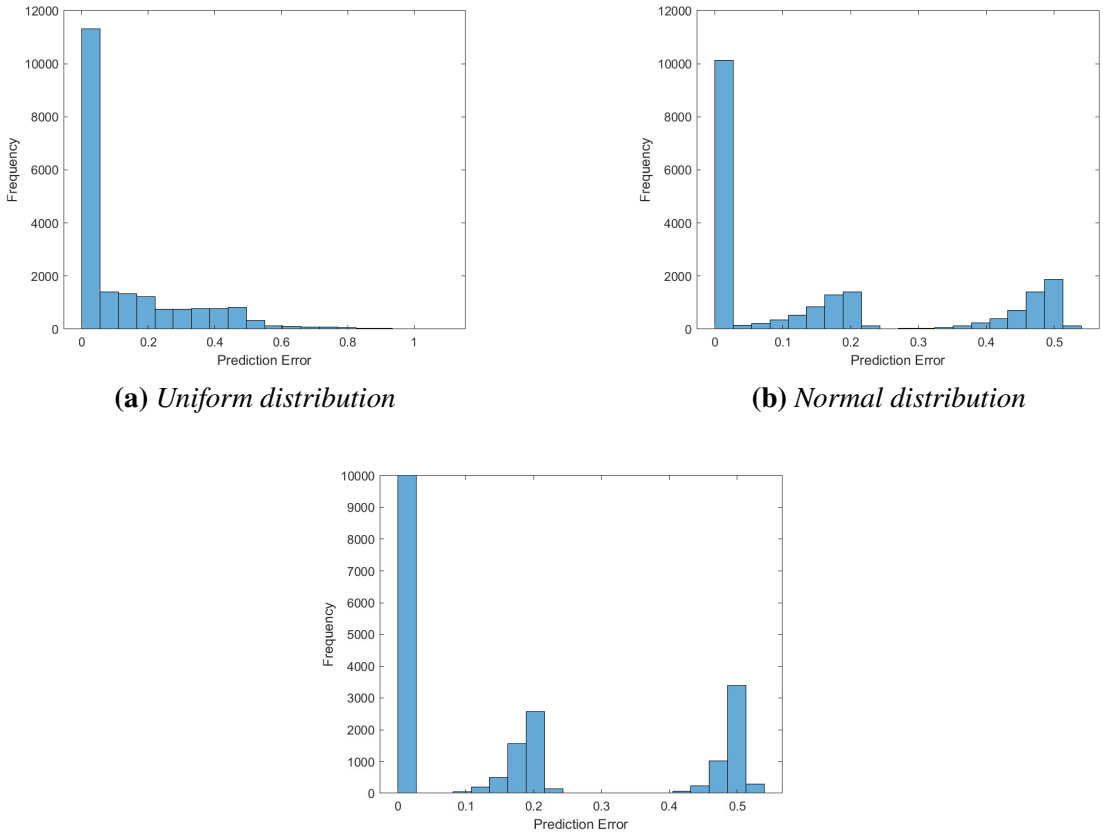


(c) *Log-normal distribution*

Figure 4.9. *Error histograms of a network with four hidden layers of two nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

since the network using three layers performed the best when nodes were held constant and the network of four nodes performed the best when layers were held constant.

In this case, our predictions were much worse. On data with a uniformly distributed initial condition, our RMSE was 0.221 and our misclassification rate was 3.93×10^{-2}. On data with normally distributed initial condition, our RMSE was 0.247 and our misclassification rate was 4.67×10^{-2}. Whereas on data with log-normally distributed initial conditions, our RMSE was 0.264 and our misclassification rate was a much higher 0.110. A histogram



(a) *Uniform distribution*

(b) *Normal distribution*

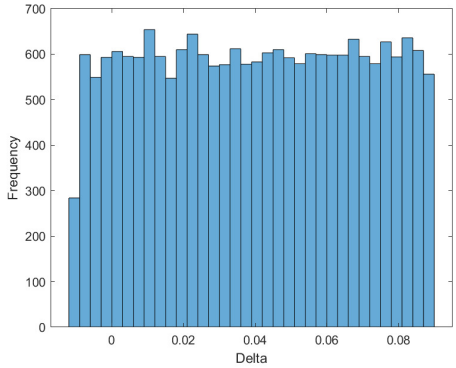
(c) *Log-normal distribution*

Figure 4.10. *Error histograms of a network with three hidden layers of four nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.*

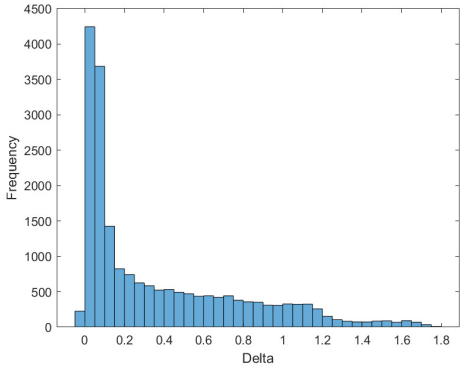
of errors is visible in Figure 4.10. Despite our poor accuracy, however, the pattern of one section of good predictions followed by two sections of progressively poorer predictions was still visible in the normal and log-normal sections.

4.4 Data Distribution of Dynamic Systems

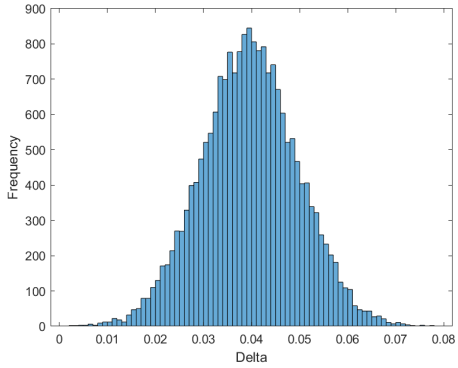
Our systems performed very well across different initial condition variation distributions. We next asked ourselves if we were able to find the cause of this high performance, and see if this was a feature of our choice of electrical microgrid fault analysis as our domain or if this pattern may be visible across all forms of distributional robustness.



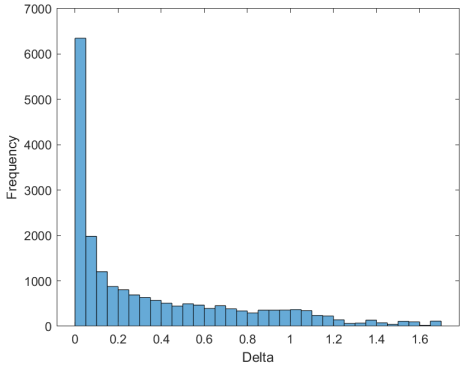
(a) *Uniform distribution at* $t = 0$



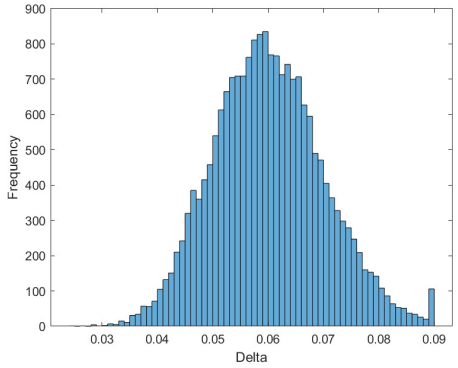
(b) *Uniform distribution at random time*



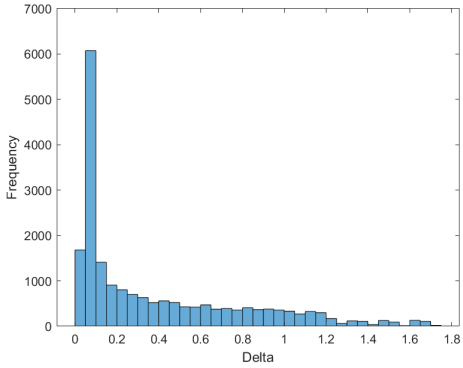
(c) *Normal distribution at* $t = 0$



(d) *Normal distribution at random time*



(e) *Log-normal distribution at* $t = 0$



(f) *Log-normal distribution at random time*

Figure 4.11. *Histogram of* $20,000$ *potential* δ_1 *choices at* $t = 0$ *and at a randomly selected time, with uniform, normal, and log-normal distributions.*

Our conjecture is that our high performance is because our data point distribution is dominated by the dynamics of the system, not the initial state distribution. As a reminder, we generate a five-second trajectory for each case, and then only analyze a random one-second window of data from all three generators with our neural network. This could mean that our system has a chance to reach a steady state when we sample our one-second window of data.

To see what this would look like, we analyzed our initial condition variation for the first generator δ_1 at both the initial time $t = 0$, which would have a uniform, normal, or log-normal distribution, and at the beginning of the random one-second window. Results are included in Figure 4.11.

At $t = 0$, our initial condition variation was predictably centered around the equilibrium for δ_1, which was 0.0396. But our randomly selected beginning of our windows of analysis were almost identical, with a large number of samples at the equilibrium itself and the rest following a similar pattern across all data distributions. This is due to the laws of dynamics in our system, which force a certain behavior on our blade as time progresses.

Still, changing the distribution in Section 4.2 did have a noticeable influence on accuracy, visible in our error histograms. In other domains less dominated by dynamics, it is likely that the effects of distributional robustness could be even more visible in other applications of machine learning.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions

5.1 Analysis

This project aimed to introduce a new form of robustness for neural network users to consider in the form of distributional robustness. Using faults in electrical microgrids as our domain, we were able to see the impact of different distributions and network sizes on distributional robustness and performance.

First, we achieved acceptable performance from a network composed of four hidden layers of eight nodes each. While wider networks did have slightly better performance, it was not computationally worthwhile to expand the network. Table 4.2 suggests an upper triangular structure to this problem, where an increase in network depth should be paired with an increase to network width to avoid an increase in misclassification rate.

In Section 4.2, we were able to see the impact of changing distributions on network performance. As our initial condition variation for our generator rotor angular position moved further from the variation we used for training, performance slightly decreased. While our misclassification rates stayed low, Figures 4.3 and 4.4 showed a decrease in accuracy over Figure 4.2a, which was the error histogram of performance on the distribution on which our network was trained.

Additionally, we were able to see the impact of network hyperparameters on distributional robustness in Section 4.3. Reducing the depth of the network while holding the width constant led to a much more significant impact on accuracy than reducing the width while holding the depth constant. Combining depth and width variations, however, led to a compounding effect on robustness, visible in Figure 4.10.

An interesting observation comes from the shape of the error histograms across changing distributions and network sizes. In almost all cases, our histograms are a similar shape, with only the scale of the error changes. Regardless of the initial condition distribution or network size used, our networks predict no fault and Fault 1 cases with a very high degree

of accuracy, followed by Faults 2 and 3 with increasing error. It would be interesting to see if the similarities in error patterns persist in different applications of machine learning, or if these are unique to power system applications.

Together, these results suggest that distributional robustness gains will be made through a series of trade-offs, similar to other forms of robustness, and must be considered when applying neural networks in real-world scenarios.

5.2 Implications and Limitations

One major limitation to this thesis comes from the dynamic nature of our application case. In our microgrid system, the laws of dynamics are much more important than our initial condition variation of rotor angle. Our distributions converge due to the convergence of dynamics. This feature is amplified by our sampling method. By taking a random one-second window of the whole five-second trajectory, we are much more likely to sample similar looking data samples, since we are giving the rotor angle a chance to approach a steady-state equilibrium. Indeed, while the initial distributions for δ in our differential equations are vastly different, Figure 4.11 demonstrates that the dynamic nature of our microgrid causes most of our trajectories to begin very near the equilibrium of 0.0396 instead of depending on the initial condition distribution. If we were applying a neural network to a real-world microgrid or other dynamics-dominated scenario, this could be used as an advantage instead of a weakness, since our network would be mostly accurate regardless of distributional changes. It is, however, worth noting that this feature will not be visible in all other domains.

Another limitation of this thesis comes from the nature of machine learning itself. Currently, the field of machine learning is not a deterministic science, depending instead on the choices of tool and the domain on which they are applied. Because of this, this thesis is only able to recommend the study of distributional robustness in other cases, not present a strong conclusion on the effect of distributional changes on machine learning performance.

5.3 Potential for Future Work

Future study of machine learning robustness may focus on real-world data generated from actual microgrids instead of simulated data. Actual changes to initial conditions could see if the dynamic domination of initial condition distribution visible in Figure 4.11 is a product of our simulations or a feature visible in real microgrids.

Additionally, future work could study the consideration of distributional robustness in other fields besides electrical microgrids. It could be useful to see if the upper triangular nature of Table 4.2 and the consistent pattern visible across our error histograms in Sections 4.2 and 4.3 are products of our microgrid application, choice of feed-forward neural networks as a ML system, or a feature visible in other studies of distributional robustness.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: MATLAB Code

A.1 Simulation Code

A.1.1 Uniform Distribution

This MATLAB code generates trajectories for data with uniformly distributed initial conditions around the each generator's equilibrium.

```
    clear all
%%%%%% parameters %%%%%%%
BaseMVA=100;
Di=[0;0;0];
H2=[47.28;12.8;6.02];
wR=2*pi*60;
Ei=[1.0566;1.0502;1.0170];
YB0R=[0.8455 0.2871 0.2096;0.2871 0.4200 0.2133;0.2096 0.2133 0.2770];
YB0I=[-2.9883 1.5129 1.2256;1.5129 -2.7239 1.0879;1.2256 1.0879 -2.3681];
Pm=[71.6;163;85]./BaseMVA;

%%%%%%%%%%

dt=1/10;
Nt=50; %the number of steps in each trajectory
t0=0;
tf=t0+Nt*dt;
w_eq=[wR;wR;wR]; %equilibrium
dlt_eq=pi/180*[2.2717;19.7315;13.1752]; %equilibriam

Nsample=5000; %number of sample trajectories
NN=Nt+1; %total number of time steps, including initial time t0
Ssample=zeros(NN,7,Nsample); %trajecotories 1:NN rows is a trajectory,
1-3 columns are w, 4-6 columns are delta, 7th column is time t
```

```
for isample=1:Nsample
    w_perturb=[0;0;0]; %initial value variation
    dlt_perturb=0.05*(2*rand(3,1)-1); %initial value variation
    wdlt0=[w_eq+w_perturb;dlt_eq+dlt_perturb];   %initial condition

    YB0R1=YB0R;
    YB0I1=YB0I;
    r=-0.05; %variation of Y matrix caused by fault
    i1=1; %fault 3
    i2=3;
    YB0R1(i1,i2)=YB0R(i1,i2)+r*YB0R(i1,i2);
    YB0R1(i2,i1)=YB0R1(i1,i2);
    YB0I1(i1,i2)=YB0I(i1,i2)+r*YB0I(i1,i2);
    YB0I1(i2,i1)=YB0I1(i1,i2);

    hndl1=@(t,wdlt) fundyn(t,wdlt,H2,Di,wR,Pm,Ei,YB0R1,YB0I1);
    %Power system model
    options=odeset('RelTol',10^(-6),'AbsTol',10^(-6));
    [tt2 wdlt2]=ode45(hndl1,t0:dt:tf,wdlt0,options); %with fault

    Ssample(:,:,isample)=[wdlt2 tt2];
end

close all
kk=randi(Nsample,1);
tt=Ssample(:,end,kk);
figure,plot(tt,Ssample(:,1,kk),tt,Ssample(:,2,kk),tt,Ssample(:,3,kk))

save data_trajectories_fault3_val

Ssample=zeros(NN,7,Nsample); %trajecotories 1:NN rows is a trajectory,
1-3 columns are w, 4-6 columns are delta, 7th column is time t
for isample=1:Nsample
    w_perturb=[0;0;0]; %initial value variation
    dlt_perturb=0.05*(2*rand(3,1)-1); %initial value variation
```

```
    wdlt0=[w_eq+w_perturb;dlt_eq+dlt_perturb];   %initial condition

    YB0R1=YB0R;
    YB0I1=YB0I;
    r=-0.05;
    i1=1; %fault 1
    i2=2;
    YB0R1(i1,i2)=YB0R(i1,i2)+r*YB0R(i1,i2);
    YB0R1(i2,i1)=YB0R1(i1,i2);
    YB0I1(i1,i2)=YB0I(i1,i2)+r*YB0I(i1,i2);
    YB0I1(i2,i1)=YB0I1(i1,i2);

    hndl1=@(t,wdlt) fundyn(t,wdlt,H2,Di,wR,Pm,Ei,YB0R1,YB0I1);
    %Power system model
    options=odeset('RelTol',10^(-6),'AbsTol',10^(-6));
    [tt2 wdlt2]=ode45(hndl1,t0:dt:tf,wdlt0,options); %with fault
    Ssample(:,:,isample)=[wdlt2 tt2];
end

kk=randi(Nsample,1);
tt=Ssample(:,end,kk);
figure,plot(tt,Ssample(:,1,kk),tt,Ssample(:,2,kk),tt,Ssample(:,3,kk))

save data_trajectories_fault1_val

Ssample=zeros(NN,7,Nsample); %trajecotories 1:NN rows is a trajectory,
1-3 columns are w, 4-6 columns are delta, 7th column is time t
for isample=1:Nsample
    w_perturb=[0;0;0]; %initial value variation
    dlt_perturb=0.05*(2*rand(3,1)-1); %initial value variation
    wdlt0=[w_eq+w_perturb;dlt_eq+dlt_perturb];   %initial condition

    YB0R1=YB0R;
    YB0I1=YB0I;
    r=-0.05;
```

```
    i1=3; %fault 2
    i2=2;
    YB0R1(i1,i2)=YB0R(i1,i2)+r*YB0R(i1,i2);
    YB0R1(i2,i1)=YB0R1(i1,i2);
    YB0I1(i1,i2)=YB0I(i1,i2)+r*YB0I(i1,i2);
    YB0I1(i2,i1)=YB0I1(i1,i2);
    hndl1=@(t,wdlt) fundyn(t,wdlt,H2,Di,wR,Pm,Ei,YB0R1,YB0I1);
    %Power system model

    [tt2 wdlt2]=ode45(hndl1,t0:dt:tf,wdlt0,options); %with fault
    Ssample(:,:,isample)=[wdlt2 tt2];
end

kk=randi(Nsample,1);
tt=Ssample(:,end,kk);
figure,plot(tt,Ssample(:,1,kk),tt,Ssample(:,2,kk),tt,Ssample(:,3,kk))

save data_trajectories_fault2_val

Ssample=zeros(NN,7,Nsample); %trajecotories 1:NN rows is a trajectory,
1-3 columns are w, 4-6 columns are delta, 7th column is time t
for isample=1:Nsample
    w_perturb=[0;0;0]; %initial value variation
    dlt_perturb=0.05*(2*rand(3,1)-1); %initial value variation
    wdlt0=[w_eq+w_perturb;dlt_eq+dlt_perturb];   %initial condition

    YB0R1=YB0R;
    YB0I1=YB0I;
    r=0.0; %if 0, there is no fault
    i1=3; %no fault
    i2=2;
    YB0R1(i1,i2)=YB0R(i1,i2)+r*YB0R(i1,i2);
    YB0R1(i2,i1)=YB0R1(i1,i2);
    YB0I1(i1,i2)=YB0I(i1,i2)+r*YB0I(i1,i2);
    YB0I1(i2,i1)=YB0I1(i1,i2);
```

```
    hndl1=@(t,wdlt) fundyn(t,wdlt,H2,Di,wR,Pm,Ei,YB0R1,YB0I1);
    %Power system model

    [tt2 wdlt2]=ode45(hndl1,t0:dt:tf,wdlt0,options); %with fault
    Ssample(:,:,isample)=[wdlt2 tt2];
end

kk=randi(Nsample,1);
tt=Ssample(:,end,kk);
figure,plot(tt,Ssample(:,1,kk),tt,Ssample(:,2,kk),tt,Ssample(:,3,kk))

save data_trajectories_normal_val
```

A.2 Neural Network Training

This MATLAB code trains neural networks built from one to five hidden layers consisting of one to sixteen nodes each.

```
clear all

n_layers=1:5;
nodes_per_layer=1:16;
train_results=zeros([5,16]);
train_results=train_results+1;
train_proportion_wrong=train_results;

val_results=train_results;
val_proportion_wrong=train_results;

for layer=n_layers
    for node=nodes_per_layer

fprintf("Start %d layers and %d nodes \n", layer, node)
```

```
%

 TRAINING DATA

X=[];
Y=[];

 Error=0
load('train_data_trajectories_normal_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+0;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

 Error=1
load('train_data_trajectories_fault1_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+1 ;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

 Error=2
load('train_data_trajectories_fault2_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
```

```
y=zeros(1,n_samples)+2;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

 Error=3
load('train_data_trajectories_fault3_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+3;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

XY=[X;Y];
XY=XY(:,randperm(size(XY,2)));
X=XY(1:end-1, :);
Y=XY(end,:);

lbX_train=min(X,[],"all");
lbY_train=min(Y,[],"all");
ubX_train=max(X,[],"all");
ubY_train=max(Y,[],"all");

a=2.0*(X - lbX_train)./(ubX_train - lbX_train) - 1.0;

setup=zeros(1,layer);
setup=setup+node;
net=feedforwardnet(setup);
```

```
net.trainParam.epochs=10000;

net=train(net,a,Y);
yy=sim(net,a);
RMSE = rms(yy - Y);
RMSE_rounded=rms(round(yy) - Y);
one_if_wrong=round(yy)~=Y;
sum_wrong=sum(one_if_wrong);
proportion_wrong=sum_wrong/length(one_if_wrong);

train_results(layer, node)=RMSE;
train_proportion_wrong(layer,node)=proportion_wrong;

network_save_name=sprintf('%d_layer_%d_node_nn.mat',layer,node);
save(network_save_name,"net")

fprintf("Train RMSE: %9.8f \n
Train proportion wrong: %9.8f \n ",RMSE, proportion_wrong)

%
 Validation data

X=[];
Y=[];

 Error=0
load('data_trajectories_normal_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+0;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
```

```
Y=[Y y];

 Error=1
load('data_trajectories_fault1_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+1 ;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

 Error=2
load('data_trajectories_fault2_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+2;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
X=[X x];
Y=[Y y];

 Error=3
load('data_trajectories_fault3_val.mat')
n_samples=size(Ssample,3);
x=zeros(30,n_samples);
y=zeros(1,n_samples)+3;
for i=1: n_samples
    r=randi(42);
    x(:,i)=[Ssample(r:r+9,1,i) ; Ssample(r:r+9,2,i) ; Ssample(r:r+9,3,i) ];
end
```

```
X=[X x];
Y=[Y y];

XY=[X;Y];
XY=XY(:,randperm(size(XY,2)));
X=XY(1:end-1, :);
Y=XY(end,:);
X_val=X;

a=2.0*(X_val - lbX_train)./(ubX_train - lbX_train) - 1.0;
yy=sim(net,a);
RMSE = rms(yy - Y);
RMSE_rounded=rms(round(yy) - Y);
one_if_wrong=round(yy)~=Y;
sum_wrong=sum(one_if_wrong);
proportion_wrong=sum_wrong/length(one_if_wrong);

val_results(layer, node)=RMSE;
val_proportion_wrong(layer,node)=proportion_wrong;

network_save_name=sprintf('%d_layer_%d_node_nn.mat',layer,node);
save(network_save_name,"net")

fprintf("Validation RMSE: %9.8f \n
Validation proportion wrong: %9.8f \n ",RMSE, proportion_wrong)

end
end
```

List of References

[1] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "UFLDL tutorial," *Chapters available at http://deeplearningstanford. edu/wiki/index. php/UFLDL_Tutorial*, 2012.

[2] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural networks*, vol. 2, no. 6, pp. 459–473, 1989.

[3] N. Lu, S. Mabu, T. Wang, and K. Hirasawa, "An efficient class association rule[U+2010]pruning method for unified intrusion detection system using genetic algorithm," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 8, March 2013.

[4] W. Yu and J. Cao, "Stability and Hopf bifurcation analysis on a four-neuron bam neural network with time delays," *Physics Letters A*, vol. 351, no. 1-2, pp. 64–78, 2006.

[5] T. Caelli, W. F. Bischof, and W. F. Bischof, *Machine Learning and Image Interpretation*. New York, NY, USA: Springer Science & Business Media, 1997.

[6] M. Barreno, B. Nelson, A. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, pp. 121–148, November 2010.

[7] O. Linda, T. Vollmer, and M. Manic, "Neural network based intrusion detection system for critical infrastructures," *Proceedings of the International Joint Conference on Neural Networks*, pp. 1827–1834, June 2009.

[8] E. Anthi, L. Williams, M. Rhode, P. Burnap, and A. Wedgbury, "Adversarial attacks on machine learning cybersecurity defences in industrial control systems," *Journal of Information Security and Applications*, vol. 58, p. 102717, 2021. Available: https://www.sciencedirect.com/science/article/pii/S2214212620308607

[9] R. Gao and A. J. Kleywegt, "Distributionally robust stochastic optimization with wasserstein distance," 2016. Available: https://arxiv.org/abs/1604.02199

[10] S. S. Vallender, "Calculation of the wasserstein distance between probability distributions on the line," *Theory of Probability & Its Applications*, vol. 18, no. 4, pp. 784–786, 1974. Available: https://doi.org/10.1137/1118101

[11] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[12] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, "Enhancing robustness of ma-chine learning systems via data transformations," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–5.

[13] H. Lin, K. Sun, Z.-H. Tan, C. Liu, J. M. Guerrero, and J. C. Vasquez, "Adaptive protection combined with machine learning for microgrids," *IET Generation, Transmission & Distribution*, vol. 13, no. 6, pp. 770–779, 2019.

[14] S. Rahman Fahim, S. K. Sarker, S. M. Muyeen, M. R. I. Sheikh, and S. K. Das, "Mi-crogrid fault detection and classification: Machine learning based approach, com-parison, and reviews," *Energies*, vol. 13, no. 13, 2020. Available: https:// www.mdpi. com/1996-1073/13/13/3460

[15] I. Almutairy and M. Alluhaidan, "Fault diagnosis based approach to protecting dc microgrid using machine learning technique," *Procedia Computer Science*, vol. 114, pp. 449–456, 2017.

[16] P. M. Anderson and A. Fouad, *Power System Control and Stability*. Piscataway, NJ, USA: IEEE Press, 2003.

[17] K. Sun, J. Qi, and W. Kang, "Power system observability and dynamic state estimation for stability monitoring using synchrophasor measurements," *Control Engineer-ing Practice*, vol. 53, February 2016.

[18] V. Vittal, J. D. McCalley, P. M. Anderson, and A. Fouad, *Power System Control and Stability*. Hoboken, NJ, USA: John Wiley & Sons, 2019.

[19] N. Kolbe, "Wasserstein distance code for matlab," 2022. Available: https://github. com/nklb/wasserstein-distance

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California