NPS-MA-23-001

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Robustness and Vulnerability Measurement of Deep Learning Methods for Cyber Defense

by

Thor Martinsen, Wei Kang, Elana Kozak, and Philip Smith

December 2022

Distribution Statement A: Approved for public release. Distribution is unlimited.

Prepared for: Navy Cyber Defense Operations Command. This research is supported by funding from the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098). NRP Project ID: NPS-22-N336-A

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE:	3. DATES COVERED:	
December 2022	Technical Report	START DATE: January 1, 2022	END DATE: December 31, 2022

4. TITLE AND SUBTITLE:
Robustness and Vulnerability Measurement of Deep Learning Methods for Cyber Defense

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
		0605853N/2098
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
NPS-22-N336-A		

6. AUTHOR(S)
Thor Martinsen, Wei Kang, Elana Kozak and Philip Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES):
Department of Applied Mathematics
Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-MA-23-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Postgraduate School; Naval Research Program
Navy Cyber Defense Operations Command
112 Lake View Parkway
Suffolk, VA 23435

10. SPONSOR/MONITOR'S ACRONYM(S):
NRP; NCDOC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NPS-MA-23-001;
NPS-22-N336-A

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The goal of this study is to investigate mathematical concepts and quantitative measures of robustness and vulnerability of machine learning systems to adversarial data and develop computational methods capable of quantitatively evaluating the robustness and vulnerability of deep learning tools that can be applied in cybersecurity settings. The first phase of the project is a literature review. The second phase of the study is focused on robustness analysis of infrastructure cyber security. Using a microgrid power system model and learning-based fault detection as the testbed, we investigate the robustness of neural networks subjected to noisy or poisoned data. Finally, the third phase of the project, explores distributional robustness. Neural networks may sometimes be used outside of the environment in which they were trained. If the distribution of the incoming data is significantly different from that of the training data, it could negatively impact the performance of the neural network. In addition to a quantitative analysis of robustness, the study reveals an underlying relationship between the robustness and the dynamical behavior the training data.

15. SUBJECT TERMS
Deep learning, robustness, cyber security, fault detection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	UU	72

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
CAPT Thor Martinsen, USN	*(831)656-2581*

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ann E. Rondeau President	Scott Gartner Provost

The report entitled "Robustness and Vulnerability Measurement of Deep Learning Methods for Cyber Defense" was prepared for the Navy Cyber Defense Operations Command and is funded by the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098).

This report was prepared by:

Thor Martinsen CAPT, USN, PhD	Wei Kang Professor

Reviewed by:	**Released by:**
Frank Giraldo, Chairman Department of Applied Mathematics	Kevin B. Smith Vice Provost for Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The goal of this study is to investigate mathematical concepts and quantitative measures of robustness and vulnerability of machine learning systems to adversarial data, and develop computational methods capable of quantitatively evaluating the robustness and vulnerability of deep learning tools that can be applied in cybersecurity settings. The first phase of the project is a literature review. The second phase of the study is focused on robustness analysis of infrastructure cyber security. Using a microgrid power system model and learning-based fault detection as the testbed, we investigate the robustness of neural networks subjected to noisy or poisoned data. Finally, the third phase of the project, explores distributional robustness. Neural networks may sometimes be used outside of the environment in which they were trained. If the distribution of the incoming data is significantly different from that of the training data, it could negatively impact the performance of the neural network. In addition to a quantitative analysis of robustness, the study reveals an underlying relationship between the robustness and the dynamical behavior of the training data.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Navy networks and infrastructure are under frequent cyberattack. One developing area of application of Artificial Intelligence (AI) and Machine Learning (ML) is cybersecurity systems. However, some machine learning weaknesses, such as lack of interpretability as well as susceptibility to data poisoning attacks, are important issues that must be studied and addressed in order to ensure the reliable and safe use of AI tools is network and cybersecurity settings. The robustness of Deep Learning (DL) techniques used in computer vision and language processing have been extensively studied. However, less is currently known about the vulnerabilities and robustness of DL methods suitable in cybersecurity applications. This report includes a literature review, case study, and computer simulations focusing on the robustness and vulnerability of machine learning methods applicable to both department of defense (DoD) and civilian applications. The work was carried in calendar year 2022 by two faculty members and two graduate students assigned to the department of Applied Mathematics at the Naval Postgraduate School. The study was funded by the Navy Research Program. Some of the work contained herein can also be found in the recently published master's degree theses of the two project students.

Sophisticated cyber actors and nation-states are developing capabilities to disrupt, destroy, or threaten the delivery of essential services. According to the Cybersecurity & Infrastructure Security Agency (cisa.gov) "As information technology becomes increasingly integrated with physical infrastructure operations, there is increased risk for wide scale or high-consequence events that could cause harm or disrupt services upon which our economy and the daily lives of millions of Americans depend" (https://www.cisa.gov/cybersecurity). Many of the United States Navy's operations depend upon physical infrastructure such as the power grid and computer networks. These days, cybersecurity plays an essential role in protecting a wide spectrum of critical systems and infrastructures. For example, the reliability of power grids depends upon robust cybersecurity protections (GAO@100 (2021)). The use of AI and Machine Learning in cybersecurity settings is a developing area that is attracting increased

attention. For instance, researchers have studied the use of supervised learning to classify particular security problems such as denial-of-service attacks or to identify different classes of network attacks such as scanning and spoofing (Berman at al. (2019); GAO@100 (2021); Sarker et al. (2020)). Many approaches to network intrusion detection using DL such as Deep Belief Networks (DBNs) and Restricted Boltzmann Machines (RBMs) (Alrawashdeh and Purdy (2016)) have also been proposed. DBNs can also be applied to detect malware attacks. Studies show that a combination of feature selection and Deep Neural Networks (DNNs) can perform tasks of malware classification (Berman at al. (2019), Sarker at el. (2020)). The stacked auto-encoder, a special kind of DNN, can be used for network traffic identification and protocol classification (Berman at al. (2019)). Despite many promising applications of DL, neural nets have been shown to be susceptible to adversarial data, such as small perturbations to the input data which cause significant variation in the output of the network leading to errors such as mislabeling. The problem of quantitatively measuring the robustness and vulnerability of DL used in cybersecurity applications is a relatively new area that requires more study.

II. LITERATURE REVIEW

A. ROBUSTNESS UNDER DATA UNCERTAINTIES OR ADVERSARIAL ATTACK

Robustness in machine learning often takes on different meanings. One definition of robustness is "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" (O'Mahony et al. (2004)). In other words, it is the ability of a Machine Learning (ML) model to perform well against noisy or adversarial generated data. In O'Mahony et al. (2004), the authors focus on an adversarial attack and define the Stability of Prediction, SOP, as a quantitative robustness measurement value. SOP measures the success of an attack, which is correlated to the robustness of the machine learning system. An SOP value close to one indicates very little change between the pre- and post-attack predictions, meaning that the system is stable, whereas an SOP value equal to zero means that the prediction changed by at least some quantity α. In this case, the system is not very robust. For example, an SOP of 0.4 at $\alpha = 2$ means that 60% of all predictions were changed by at least $+2$ units. One advantage of the SOP measurement is that it can be used to compare the robustness of various recommendation systems, regardless of their individual accuracy.

Hembram et. al. studied robustness in the context of a soil erosion, using a gully erosion prediction model. In their research, they defined robustness as a combination of discrimination ability and reliability (Hembram et al. (2021)). The discrimination ability of the system referred to the ability of a model to separate between a gully presence and absence areas, whereas reliability was the accuracy of the predicted gully locations compared to the observed locations. Their analysis provided several methods of measuring both aspects of robustness. For discrimination ability, they used efficiency, the Jaccard index, the Matthew's correlation coefficient, the Kappa coefficient, as well as receiver operating characteristics. For reliability they calculated the root mean square error (RMSE) and the mean absolute error (MAE). These common statistics equations can be used to compare a variety of ML models. As shown in the gully erosion study,

combined they can produce a detailed picture of the ML predictions, however, each measurement focuses solely on one aspect of the problem. Most commonly, the RMSE is used as a benchmark error measurement.

Derks et. al. (1995) performed a robustness analysis on two different types of neural networks, namely the Radial Base Function (RBF) and the Multi-layered Feed-forward (MLF) network. In their research, they generated noisy data sets by sampling noise from a normal distribution such that the error ratio was around 1%. They subsequently trained both types of networks on the original and modified data sets. Using the root mean squared error, root mean squared error of prediction, and percentage explained variance, they concluded that the RBF model was more robust. This research demonstrates how noise can be added to input data and suggests ways of measuring its effect. While their results were specific to the data sets they used, their methods could be applied to other ML models in order to test robustness.

B. ROBUSTNESS ACHIEVED THROUGH TRAINING

Developing training methods to achieve robustness is also studied in Robey et al. (2020). Given a classification task with data drawn from a joint distribution $(x, y) \sim D$, with $x \in \mathbb{R}^d$ and $x \in \{0,1, \dots, k\}$, and a loss function $l(x, y; w)$, the goal of a machine learning algorithm is to find the weights, w , which minimize the risk over the distribution D. This can be written as

$$\min_{w} \mathbb{E}_{(x,y)\sim D} [l(x, y; w)]$$

A network using this training function is vulnerable to adversarial attacks, meaning a new input x^{adv} is close to the original value x with label y, but the predicted class of x^{adv} is not y. Perturbation-based robust learning is used to train neural networks "to be robust against a worst-case perturbation of each instance x" (Robey et al., 2020). This is formulated as a min-max problem, where the goal is to minimize the risk over D of a maximized perturbation $\delta \in \Delta$.

$$\min_{w} \mathbb{E}_{(x,y)\sim D} [\max_{\delta \in \Delta} l(x + \delta, y; w)]$$

The problem with adversarial training is that this method often fails to protect against natural variation in the data. Rather than focusing on adversarial attacks, Robey et. al. propose another method, namely model-based robust deep learning, to account for this natural variation. The focus of their study is image classification, so the examples are natural variation such as snowy conditions or background color. This method of model-based robustness requires a model of natural variation, $G(x, \delta)$, which is a mapping that takes input datum x and a nuisance parameter δ to a naturally varied output x' (Robey et al., 2020). Finding this model of natural variation can be difficult since their geometry is often more complex than that of adversarial perturbations. In fact, the perturbation-based adversarial training method is a special case of the model-based method, where $(x, \delta) = x + \delta$ for $\delta \in \Delta \coloneqq \{\delta \in \mathbb{R}^d : ||\delta||_p \leq \varepsilon\}$. For image classification problems, a known model of natural variation for rotation is $G(x, \delta) = R(\delta)$ for $\delta \in \Delta \coloneqq [0, 2\pi]$. If the model of natural variation is not *a priori* knowledge, then $G(x, \delta)$ should be learned from the data before starting the robustness training. This is done by separating the data into two sets *A* and *B*. The set *A* contains the original data and the set *B* contains the data with natural variation. The goal is then to find a model G which transforms the distribution of data in *A* into the distribution of data in *B*.

The model-based robustness training algorithm is similar to the adversarial perturbation-based method. It is formulated as a min-max problem. This optimization problem is often difficult to solve exactly, but the problem can be modified to a finite-sample setting which is relatively easy to solve. These model-based robustness methods were evaluated by comparing the standard training model with no robustness considerations to an adversarial training model with the perturbation-based method. The results in Robey et al. (2020) show that the model-based methods provide significant robustness improvements to a variety of datasets and nuisances. Additionally, model-based training provides an advantage over other methods, even when tested on datasets with higher natural variation than the set it was trained on. Overall, their model-based robust deep learning techniques show significant improvement over other methods when used in image classification problems.

The idea of defining a model of variation may be useful for other types of data sets, apart from image classification problems. This approach can help train networks against many types of predictable noise. The main challenge with this approach is defining the model. Once an accurate model of variation is found, any of the three model-based training methods would provide significant robustness improvement.

C. INFRASTRUCTURE CYBER SECURITY

Implementing cybersecurity defenses within infrastructure such as power systems is increasingly important. Power systems are relevant in nearly every aspect of modern life, and we rely upon them being resilient despite their complexity. Nevertheless, disturbances from natural events, maintenance, or even attacks can occur. Determining the location and causes of disturbances is usually done by experienced human operators, however, machine learning has recently been applied to differentiate between types of disturbances (Hink et al. (2014)). Hink et. al. tested various machine learning methods on a set of simulated data from Mississippi State University which contained five types of disturbances: short-circuit fault, line maintenance, remote tripping command injection (attack), relay setting change (attack), and data injection (attack). Their goal was to classify the disturbances into three classification schemes, with either 37 specific event scenarios, three types of events (attack, natural event, or no event), or a binary scheme of attacks and normal operations. The three-class scheme with a *JRipper + Adaboost* method produced the best results, proving that using machine learning is a viable approach to power system disturbance classification.

Another study conducted in 2019 by Wang et al. used data from Phasor Measurement Units (PMU) within the power system to classify disturbances. Like the Hink research, this study also used the three event classes: no event, natural event, and intrusion event. To improve flexibility and gain higher accuracy and robustness in this model, the researchers started by ”manually constructing new features from the original data in the dataset so as to enhance data dimension.” Subsequently, they performed data splitting and training by sorting the original features according to importance and then selecting diverse proportions of those features. This reduced the error caused by bad PMU

measurements. Finally, their model assigned different weights to each label and chose the final classification label based on these weights. The model was tested on a dataset from the ICS cyber-attack datasets and success was measured in terms of accuracy, prediction, recall, F1 score, the ROC curve, and the Area under the ROC Curve (AUC). This study showed that splitting the data and using manufactured features had a positive effect on the classifier's predictions of power system disturbances. Fault detection using machine learning is proposed in Rahman et al. 2020.

D. DISTRIBUTIONAL ROBUSTNESS

Due to a variety of reasons, including operator error, adversarial influence, or simply being the best option available, neural networks are sometimes used outside of the environment in which they were trained. This may negatively impact their performance to a greater or lesser degree. Broadly speaking, the desire for model robustness includes the ability of a machine learning system to remain stable across and outside of different probability distributions. 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. Distributional robustness may also affect non-adversarial situations, such as broader applicability of a model or maintaining accuracy after an unanticipated shift in probability distributions. In this report, distributional robustness is defined as the quantifiable difference between the performance of a model on a given data distribution compared to the model's performance on data taken from the distribution upon which the model was originally trained.

Many attempts have been made to create robust machine learners and optimizers built on unknown data distributions. Oftentimes, they either rely upon known moments (such as mean and covariance) or on making worst-case estimates within a limited bound of possible data distributions. Gao and Kleywegt (2016) came up with a statistical-distance-based Distributionally Robust Stochastic Optimizer (DRSO) that allows consideration of otherwise unwieldy data distributions. By considering the Wasserstein metric (Vallender (1974)), Gao and Kleywegt were able to obtain precise structural descriptions of worst-

case distributions, allowing model creators to hedge their bets against likely and unlikely statistical distributions.

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 et al. (2019) demonstrated that self-supervision increases a model's robustness to both adversarial examples as well as dealing with near-distribution and out-of-distribution data points. 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.

Notably, a high degree of robustness is not always necessary or desired in a machine learning systems. Several papers reviewed in this study found that an increase in machine learning robustness, also brought with them a decrease in performance on the original distribution, since the model was trained to prioritize more than simple performance Bhagoji et al. (2018). Consequently, machine learning systems may require different levels of robustness depending upon the application for which they are intended.

E. OTHER RELATED LITERATURE

Machine learning applications in cybersecurity settings are numerous, and include network routing, network traffic monitoring and anomaly detection, computer intrusion detection and prevention, as well as safeguards put in place to thwart data poisoning attacks. Machine learning systems have proven to be very useful in detecting irregular network activity 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) (Lu et al. (2013)). This approach helped reduce flaws such as high false positive rates associated with misuse and anomaly detection. Symmetric key cryptography may in the future also benefit from the application of machine learning. In 2006, Yu and Cao proposed using time varying delayed chaotic Hopfield neural networks to form

pseudorandom binary strings used to encrypt large multi-media files. 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 undergoes steps to generate descriptions of objects (Caelli et al. (1997)). While both computing power and machine learning capabilities have increased in the years since Caelli and Bischof first published this book, it still forms a solid basis for understanding the machine learning applications in computer vision. Interestingly, machine learning can also be enlisted to protect machine learning systems 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 (Barreno et al. (2010)). Many issues related to machine learning system defense, such as increases in computational costs and decreases in accuracy remain, but machine learning systems may be able to assist with optimizing such factors.

Linda, Vollmer, and Manic tailored an intrusion detection system for specific applications in critical infrastructures. Using neural networks for cluster boundary modeling, their Intrusion Detection System - Neural Network Model (IDS-NNM) managed to capture all intrusion attempts using previously unseen data rather efficiently (Linda et al. (2009)). In their research, they envision applications in nuclear power plants, power systems, or other Supervisory Control and Data Acquisition (SCADA) fields, especially given the fact that neural network-based models may be able to detect new attack instances that a trained operator had never experienced before. Incorporation of such models can, however, pose additional threats. Recognizing the fact that machine learning systems may introduce additional system attack vectors, Anthi, Williams, Rhode, Burnap, and Wedgburry explored 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 (Anthi et al. (2021)).

F. SUMMARY

During the literature review, we discovered that the concept of machine learning robustness is not unique. In fact, it oftentimes is not rigorously defined. There are a variety of different aspects of robustness that may affect the performance of a machine learning model. In addition, improving machine learning system robustness could negatively impact the performance of the model. Consequently, the implementation and evaluation of robustness should be tailored with the application in mind. Machine learning applications in cybersecurity is a relatively new area of research. Most machine learning robustness studies have thus far focused on traditional machine learning applications such as computer vision, image classification, and language processing. However, some techniques found in the existing literature are applicable to a wider spectrum of application scenarios. Overall, the robustness of machine learning systems employed in infrastructure, network, and cybersecurity settings is an important area that calls for more research.

III. A CASE STUDY – MACHINE LEARNING AND ROBUSTNESS OF MICROGRID FAULT DETECTION

Before being able to fully realize the promise of machine learning, we must first ensure that we trust how such systems function throughout the full range of their expected operating environments. This in turn requires that we explore how the underlying algorithms work and gain an understanding for the limits to their reliability and robustness. The robustness of DL techniques used in computer vision and language processing have been extensively studied. However, less is currently known about the vulnerabilities and robustness of DL methods suitable in cybersecurity applications.

In machine learning, data is essential in order to train the system. Some data is random in nature and adheres to a probability distribution, while other data may be dynamical in nature and obeys a system model. Typical examples of the former are those related to computer vision and language processing. The robustness of these associated DL models has been extensively studied. Many infrastructure-related cybersecurity problems, however, are dynamic in nature and adhere to some dynamical system, which can often be modeled using first principles or learning-based approaches. The robustness of DL models trained using dynamical data is thus far underrepresented in the body of existing security related DL research. Using a microgrid system model as an example, we analyzed the robustness of various Deep Neural Networks (DNNs) under natural disturbances, operator error, or adversarial attacks. In power grids, data forms a time sequence following the trajectory that obeys the physical laws of power systems. Microgrids are a special type of power systems that are essential to DoD operations.

Microgrids are local energy grids that can be disconnected from a larger grid and operate independently (Connecticut Department of Energy and Environmental Protection (2020)). These can be used in conjunction with a larger, traditional power grid to provide stability for critical infrastructure. For machine learning, microgrids provide a prime environment to analyze the system and make predictions based on specific measurements. As with any power grid, faults can occur due to natural disturbances,

operator error, or adversarial attacks. Typically, human operators are used to determine where these issues occur, however, this task is suited for machine learning programs as well (Hink et al. (2014), Lin et al. (2019), Almutairy and Alluhaidan (2017)). Microgrids are particularly useful for studying these machine learning techniques because strategically measured data can sufficiently represent the functioning power system, thereby allowing the neural network to make accurate predictions. Despite the many applications and exciting potential of machine learning, it brings with it weaknesses that can be exploited. Neural networks are only as good as the data on which they are trained. Understanding the vulnerabilities and limitations of machine learning is crucial when incorporating these technologies into existing systems. For military applications in particular, we must be absolutely confident in the machine learning system's ability to perform correctly under adverse conditions brought about by either natural phenomena or deliberate attack. Machine learning systems must therefore be adaptable to the environment in which they operate and resistant to adversarial input. Moreover, the operators of equipment containing machine learning components must be aware of machine learning limitations and should have a general understanding of the machine learning model upon which their system was built. Just as engineers study theoretical concepts before applying them, analysts must understand how machine learning works before relying on its results. With a greater understanding of the underlying principles, operators can critically assess the real-time predictions provided by their machine learning system.

Due to its relative simplicity, our ability to control network variables, and the availability of training data, this research used a simulated microgrid called the 9-bus model (Vittal et al. (2019)). The research focused on robustness, a broad term that encompasses a system's ability to perform well in the presence of noise and uncertainties. We began by designing and training a simple feedforward neural network using MATLAB's neural network toolbox. This network served as a baseline for follow-on experiments, since it performs well under perfect conditions but incorporates no additional robustness measures. Once built, we added three types of noise (uniform random, Gaussian, and adversarial) to the testing data and measured the network's performance. We

subsequently tested other network structures and measured their performance and stability against added noise. Finally, we trained new neural networks using training data with added noise in the hopes of improving their robustness against noisy testing data. Based on these tests, we determined the optimal structure for a machine learning tool using power grid data. Analysis of the average errors over large testing data sets showed the limits of the amount of random and adversarial noise that could be added while maintaining prediction accuracy. We subsequently provide recommendations for the training and use of a machine learning tool along with parameters for which the tool can be trusted. The goal of this analysis is that it adds to our overall understanding of machine learning robustness and suggests ways in which we can quantify and measure this important characteristic. Although the neural networks developed and tested in this research were employed in a microgrid, the methods and conclusions obtained herein can be applied to more complex networks and data sets found in cybersecurity applications.

A. ROBUSTNESS – DEFINITIONS AND QUANTITATIVE MEASURES

There are many definitions of robustness for machine learning, all of which can in broad terms be summarized as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" (O'Mahony et al. (2004)). In other words, it is the ability of an ML model to perform well despite noise or adversarial data inputs. O'Mahony et al. focus on an adversarial attack and define the stability of prediction as a robustness measurement. Their definitions are given below.

Definition 3.1. (O'Mahony et al. (2004)) For each user-item pair $(a, j) \in A$, the prediction error, E_p, of prediction pre- and post-attack T is given by

$$E_p(a, j, T) = p'_{a,j} - p_{a,j}$$

where $p_{a,j}$ and $p'_{a,j}$ are pre- and post-attack predictions respectively.

Definition 3.2. (O'Mahony et al. (2004)) The stability of prediction (SOP) of the set A to an attack T is given by

$$SOP(A,T,\alpha) = 1 - \frac{1}{|A|}\sum_{a\in A}\kappa_{a,j}(\alpha)$$

where α is an arbitrary prediction shift. When $\alpha \geq 0$, $\kappa_{a,j}(\alpha) = 1$ if $E_p(a, j, T) \geq \alpha$ and 0 otherwise, when $\alpha < 0$, $\kappa_{a,j}(\alpha) = 1$ if $E_p(a, j, T) \leq \alpha$, and 0 otherwise.

The SOP value measures the success of an attack, which is correlated to the robustness of the machine learning system. A value close to 1 indicates that very few predictions changed by more than α due to the attack, meaning the system is stable. A value close to 0 means that many predictions changed by at least α, so the system is not very robust. As an example, an SOP of 0.4 at $\alpha = 2$ means that 60% of all predictions were changed by at least +2 units. If the benchmark of accuracy for this system was ±2, then an SOP of 0.4 shows it is not robust against that particular attack. These measurements of robustness are designed to test an ML model against an adversarial attack. The prediction error gives a measure of the magnitude of the change that an attack causes, whereas the SOP gives an analysis of the attack's effectiveness. One advantage to the SOP measurement is that one can use it to compare the robustness of various recommendation systems, regardless of their individual accuracy.

B. MACHINE LEARNING

Machine learning is a wide area of study whose overall goal is to use algorithms to make predictions based on input data. The applications are wide ranging, from simple classification tasks to complex game play or modeling. Within the realm of machine learning algorithms, supervised learning is a method of training a network to make predictions based on examples. This requires a large set of data which can be broken into separate training and testing sets such that the algorithm "learns" on the training data and is later validated on previously unseen testing data. The training process uses an optimization method to minimize a loss function, or difference between the predictions and actual values. This research focuses on neural networks, which take input data into a series of neurons, each with an activation function, and then output a prediction. A single neuron can be visualized with the following image.

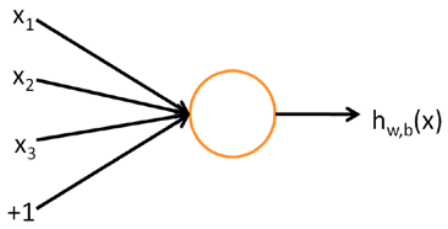


Figure 3.1. A neuron

Here the neuron takes in the inputs x_1, x_2, x_3 and a $+1$ intercept term, then outputs $h_{W,b}$ $(x) = f\ (W^T\ x + b)$ where f is the activation function, W is the set of weights represented by a matrix, and b are the bias terms (Ng et al. (2017)). Common activation functions are graphed below.

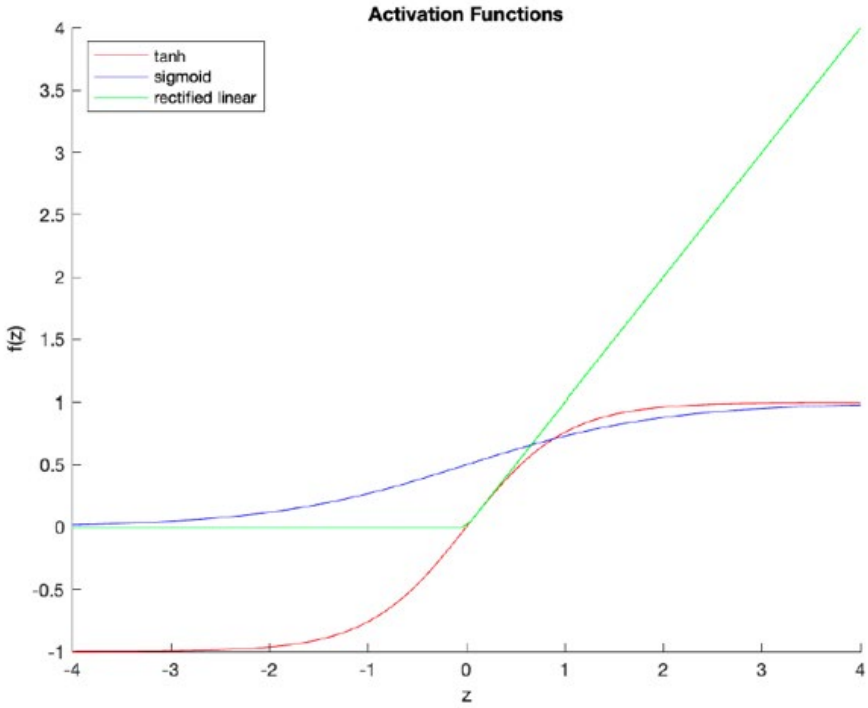


Figure 3.2. Activation functions

The sigmoid function is

$$f(z) = \frac{1}{1 + exp(-z)}.$$

The hyperbolic tangent, or “tanh” function, is

$$f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

The rectified linear (ReLU) function is

$$f(z) = max(0, z).$$

A neural network is composed of layers, each with a set of neurons. The leftmost layer is called the input layer and the rightmost is the output layer. Any layers in between are referred to as hidden layers. The example in Figure 3.3 takes in data $x \in \mathbf{R}^3$ and outputs $y \in \mathbf{R}^2$. It has two hidden layers, the first with three neurons and the second with two.

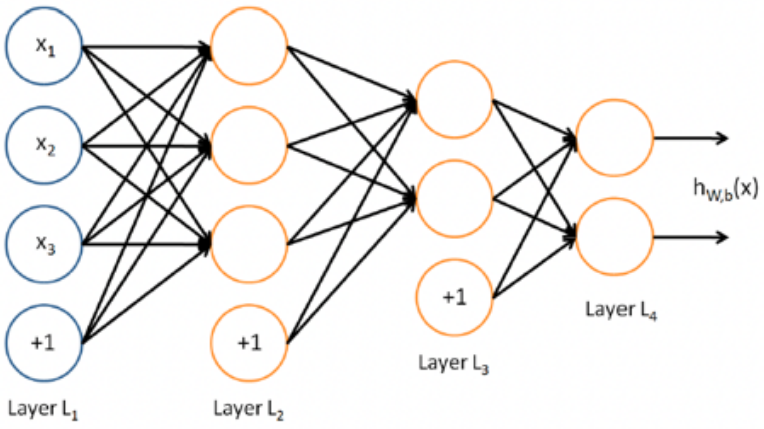


Figure 3.3. A neural network

The neural net uses parameters (W, b) where $W^{(l)}_{ij}$ denotes the weight of the connection between unit j in layer l and unit i in layer l+1. The parameter $b^{(l)}_i$ is the bias associated with unit i in layer $l + 1$. The output value of unit i in layer l, also called the activation, is denoted $a^{(l)}_i$. This is computed as $a^{(l+1)} = f\ (z^{(l)})$ where $z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$. These calculations are referred to as forward propagation. The final output, or hypothesis, is $h_{W,b} = a^{(L)}$, where L is the final layer. The final layer often uses a different activation function to produce the final prediction. For this research, we considered the pure linear and softmax functions (default for MATLAB's predefined network structures). The pure linear transfer function (called purelin in MATLAB) outputs the input (i.e., $f\ (x) = x$). The softmax function, however, outputs a vector of probabilities:

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x;\theta) \\ P(y = 2|x;\theta) \\ \cdots \\ P(y = K|x;\theta) \end{bmatrix}$$

where θ are the parameters of the model (Ng et al. (2017). This is used for classification networks where the prediction should be one of K categories. To train the neural network we used the backpropagation algorithm (Sanger (1989), Rumelhar et al. (1986)). This method uses standard gradient descent to reduce the cost function, $J\,(W, b)$. Given a training set (x^1, y^1), ..., (x^m, y^m) the cost function is computed as

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^i,y^i)\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{j,i}^l)^2$$

where

$$J(W,b;x^i,y^i) = \frac{1}{2}||h_{W,b}(x^i) - y^i||^2.$$

Recall that L is the number of layers and s_l is the number of neurons in layer $l \leq L$. The weight decay parameter λ is used to balance the two terms. Note that it is only applied to W, not the bias term b. This research focused primarily on feedforward multi-layered neural networks, also called deep neural networks. However, there are many other forms of machine learning algorithms and network structures which may be topics for further study.

C. MODEL, DATA AND DNN ROBUSTNESS ANALYSIS

1. A power system model

To train and test a machine learning method, we first require large quantities of data. We began by creating a model power grid and then gathered simulated data. Our system was based on the classical 9-bus dynamical microgrid model, as described by Anderson and Fouad (2008). It contains three generators (denoted by i= 1, 2, 3) and is modeled with "6 state variables, rotor angle (δ_i) and angular velocity (ω_i), and twelve parameters determined by the admittance." The parameters are constants, but the state variables vary

with time. The relationships between power, rotor angular velocity, degree of rotor rotation, and time are described by the following system of ordinary differential equations,

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

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

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

where H_i is the generators stored kinetic energy, ω_R is the angular velocity, ω_i is the generator's angular velocity, D_i is the drag damping effect of the generator's dynamical electrical load and system drag, and δ_i is the generators rotor angular position. The two power components are P_{mi}, the generator's mechanical energy, and P_{ei}, the produced electrical energy, with E_i as the generator's constant excitation voltage. The conductance of each generator at term i is G_{ii}. 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. More details of the model refer to Anderson and Fouad (2008). A drawing of the classical 9-bus model is shown in Figure 3.5.

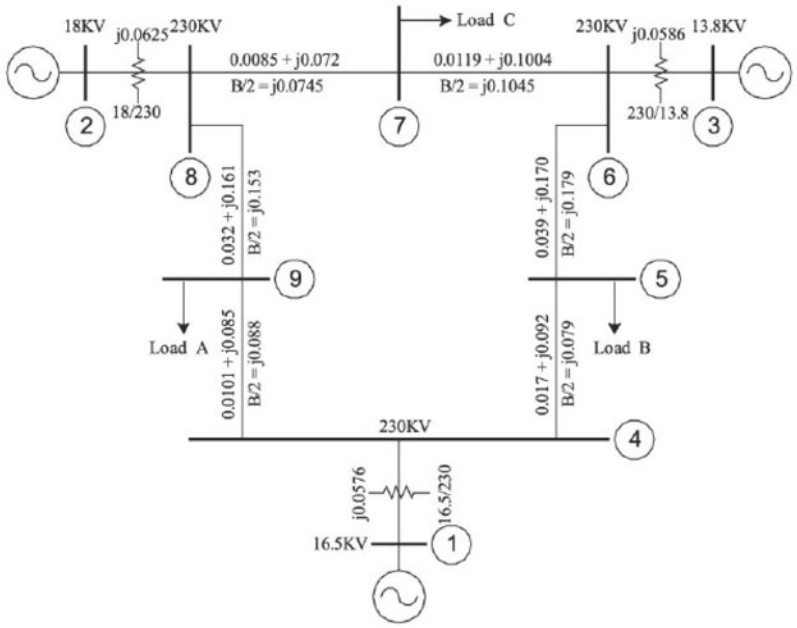


Figure 3.5. The 9-bus power system model

2. Data generated for DNN training

We implemented the 9-bus system described above in MATLAB. We then simulated trajectories for normal operations and faults at each generator. We divided the data into two sets, training and validation. Each contains 18000 examples with data $x \in \mathbf{R}^{30}$ which represents the angular velocities from each generator, measured at 10HZ, over a one-second interval. Each x corresponding to $y \in \{0, 1, 2, 3\}$ denoting the type of fault. More specifically, $y = 0$ means there is no fault (i.e., normal operations), $y = 1$ represents a fault between generators 1 and 2 which causes a 5% change in the parameters, $y = 2$ represents a fault between generators 2 and 3, and $y = 3$ represents a fault between generators 3 and 1. The data sets were split with 50% of the values representing normal operations ($y = 0$) and the remaining 50% divided evenly between the three fault locations ($y = \{1, 2, 3\}$). Once generated, the training data was used to train each neural network and the testing data was used to measure their performance.

3. Tools for DNN training

To build and train our neural networks we utilized MATLAB's NNtraintool. The simple command "network" creates a custom shallow neural network, where the user can define the number of inputs, layers, and connections. The user can subsequently train the network according to a customizable training function and training parameters. We found optimal results with the default training function, the Levenberg-Marquardt method (MathWorks Help Center - accessed 2022). Each hidden layer used the hyperbolic tangent activation function. For the setting in the training process, our baselines were 500 training iterations, a gradient of 10^{-7} and 100 validation checks. Once trained. the network, was saved so it could be used later to make predictions from different input data.

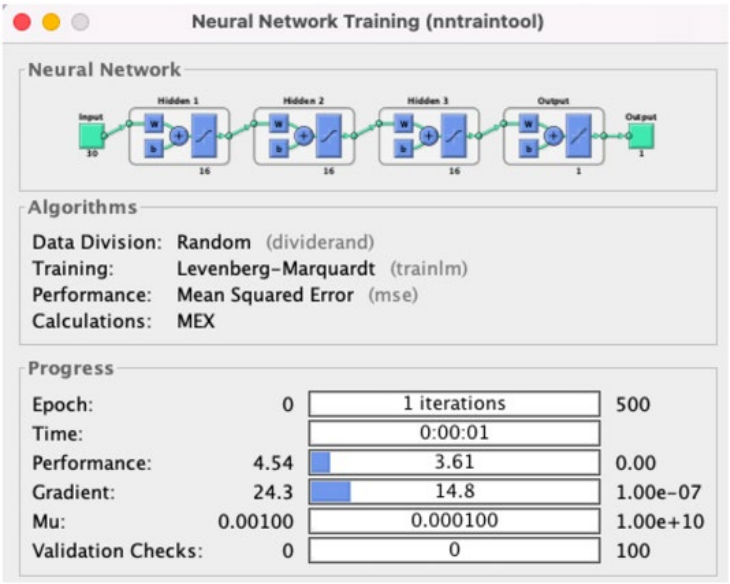


Figure 3.6. MATLAB's NNtraintool for neural networks

4. Network architecture

The neural network tool in MATLAB contains several basic network structures. The first network that we trained was a feedforward net (FFN). The number of hidden layers and nodes per layer can vary, but in all cases the input layer is connected to the first layer, which is then connected to the second layer, and so on until the final hidden layer is connected to the output layer. We tested both the feedforward net for regression and the pattern net for pattern classification. These MATLAB commands return the same network structure but are designed for different data types. The basic feedforward net is for regression, where the target values are continuous, whereas the pattern net (PN) requires target values in the form of a vector, with all zeros except a one in the index of the desired category. Additionally, both networks use the hyperbolic tangent (tansig) function for their hidden layer activations, but the feedforwardnet uses a pure linear (purelin) function for the output layer whereas the patternnet uses the softmax function for its output.

The second network we tested was the cascade forward net (CFN). This structure is similar to that of a feedforward network but includes a connection from the input and every previous layer to following layers, thereby increasing the overall number of

network connections. Each hidden layer used the hyperbolic tangent (tansig) activation function, and the output layer used the pure linear (purelin) function. As before, we customized the number of layers and nodes. It is important to note that this network takes a particularly long time to train due to its high network density.

5. Perturbations in data for robustness study

Starting with a network trained on the "clean" data (generated by the 9-bus simulation), one way to study robustness is to add various forms of noise or perturbations to our validation data set and record the new error. The two distributions we used were uniform random ($U[-\sqrt{3}, \sqrt{3}]$) and Gaussian ($N(0, 1)$). Note that these distributions have the same mean ($\mu = 0$) and standard deviation ($\sigma = 1$). A noise scalar, δ, was then multiplied by the noise and added to the normalized data. For example, the process of adding uniform random noise (Z) to a normalized data vector, $\boldsymbol{x}_{norm}$, is calculated as:

$$Z = 2\sqrt{3} \cdot rand(size(\mathbf{x}_{norm})) - \sqrt{3}$$

$$\mathbf{x}_{noise} = \mathbf{x}_{norm} + \delta \cdot Z$$

Since the networks were trained on a "clean" data set, i.e., no added noise, the "noisy" data was different from the training set and therefore posed a challenge to the robustness of the network. To further challenge the robustness of the networks, we injected validation data with noise in the direction of the gradient. This so-called gradient-perturbed data is a simple form of adversarial attack, which is intended to cause misclassification. To perform the gradient-perturbations we first wrote a script to calculate the gradient at one data point using the finite difference method:

$$[\nabla(h_{W,b})]_i(\mathbf{x}) = \frac{h_{W,b}(\mathbf{x} + he_i) - h_{W,b}(\mathbf{x} - he_i)}{2h}$$

where $h_{W,b}$ is the trained neural network and h is a very small number, typically around 10^{-5}. This was repeated for each index, i, of $\mathbf{x}$. Since our input data $\mathbf{x} \in \mathbf{R}^{30}$, this gradient $\nabla(h_{W,b}(\mathbf{x})) \in \mathbf{R}^{30}$. The noise vector ($Z$) was computed for each data point, $\mathbf{x}$, by calculating $\nabla(h_{W,b}(\mathbf{x}))$ and normalizing it. This noise was then multiplied by the noise scalar and added to the normalized data, as shown below.

$$Z = \frac{\nabla(h_{W,b}(\mathbf{x}))}{||\nabla(h_{W,b}(\mathbf{x}))||}$$

$$\mathbf{x}_{noise} = \mathbf{x}_{norm} + \delta \cdot Z$$

Computation using the finite difference method was slow. We therefore tested the gradient-perturbations on smaller data sets with $N = 1000$ test values. Prior to each test we shuffled the validation set and then selected the first N data points, to ensure representative results were obtained. Although not included in this study, automatic differentiation is another way to calculate Z as a faster algorithm for large data sets.

6. Real data

In addition to the simulated data from a 3-generator 9-bus microgrid model, we also collected real data. To do this, we set up a Frequency Disturbance Machine (FDR) as part of the University of Tennessee's FNET/GridEye project (University of Tennessee Knoxville document - accessed 2022). Our system was composed of the FDR and a small air compressor connected to the same conventional 120V power outlet. These components are shown in Figures 3.7 and 3.8. By running a program using MATLAB we collect the frequency output of the outlet over a given time interval. Two types of data were collected: normal operations and operations with load change. The load change was achieved by intermittently turning on the air compressor. In the first case, the system recorded the frequency output over three-minute intervals while nothing was energized. In the second case, data was collected in three-minute intervals with the air compressor being repeatedly turned on for 5 seconds and then off for a 10 second period. To prepare the data for use in a neural network, we randomly selected 12-second intervals and formed our input vectors, $x \in \mathbf{R}^{120}$ of the frequency output from the FDR. Each x was assigned either a value of $y = 0$ for normal operations, or $y = 1$ if the air compressor was turned on. The training and testing sets each contained 2750 examples, with about 40% representing normal operations and the rest containing air compressor activity. Feedforward neural networks were trained according to the same process outlined in Section III.B.

Figure 3.7. FDR remote sensor

Figure 3.8 Compressor used to change power load

7. Robustness measurements

There are numerous ways of measuring the robustness of a ML method. For this research we focused on the average error, stability of prediction (SOP), and percent misclassified. After each network test we first calculated the errors for each data point, i.e. the absolute value of the difference between predicted and actual Y values. The average error is simply the mean of this set. The SOP measures how much an "attack" (or noise) affects the output of a neural network (O'Mahony et al. (2004)). This was calculated using the prediction error, E_p, defined in Definitions 3.1-3.2. For our purposes we used a prediction shift $\alpha = 0.5$ since any change in prediction greater than 0.5 would result in a misclassification after rounding. The attack, T, represented the added random or gradient-based noise. The percent misclassified was used to compare the classification networks (pattern net structure) to the regression networks (feedforward and cascade forward). To do this, we rounded the regression output to the nearest integer. Then the error measurement was the percent of data points with the wrong predicted value ($y \in \{0, 1, 2, 3\}$).

D. RESULTS

1. Hyperparameter baseline

Hyperparameters that define the structure of neural networks were determined through numerical experimentations. We began by training networks of 1 to 10 layers, each with 16 nodes per layer. Uniform noise with the noise scalar $\delta = \{0, 0.01, 0.05\}$ was added to test their robustness with respect to average prediction error. The results are shown in Figure 3.9.

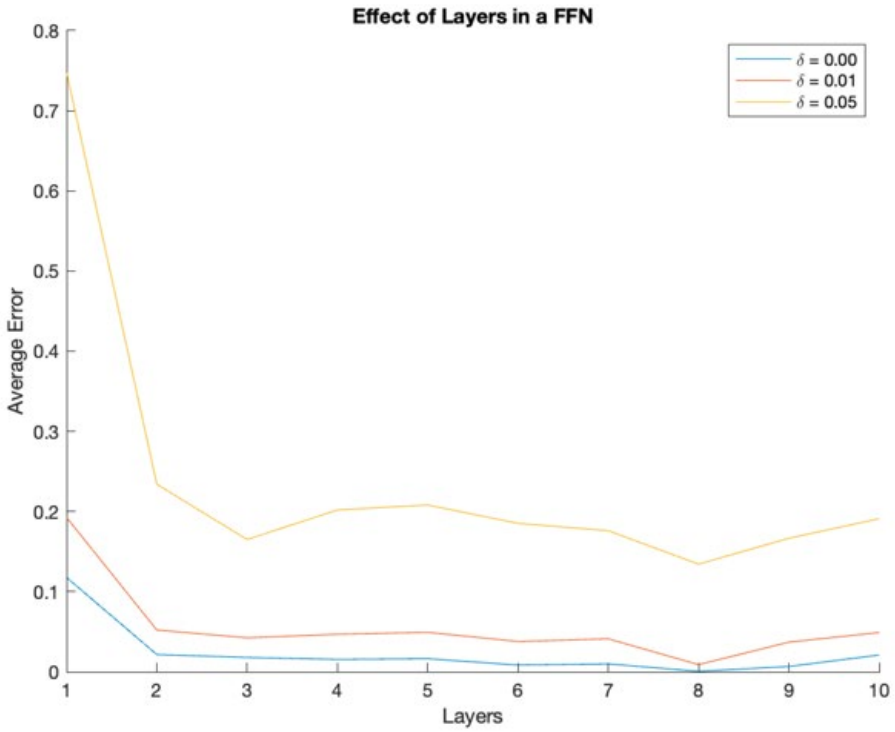


Figure 3.9. Error of FNN with 16 hidden nodes in each layer and different number of layers.

As the graph shows, the average error evened out around 3 layers regardless of the noise scalar. While there may be slight improvements for much higher numbers of layers, the time to train these networks also increased significantly. For example, using a MacBook Pro with the M1 chip (8 core CPU), a 3-layer FFN took roughly 13 minutes to train, 6 layers took 36.5 minutes, and 9 layers took 1 hour and 28 minutes. We therefore decided that 3 layers was sufficient for good results with a reasonable training time requirement. We subsequently vary the number of nodes per layer of a 3-layer FFN. We tested the networks against data with added uniform noise. Figure 3.10 shows the average error of networks with 4, 8, 12, 16, or 20 nodes per layer as the noise scalar increases. The testing results indicate that the optimal number of nodes per layer is 16. As expected, using less than 8 nodes fails to produce a very accurate or robust network. On the other hand, a greater number of nodes does not necessarily result in better performance. For instance, figure 3.10 shows that the FFN with 20 nodes performed worse than the FFN with 16 nodes. It is unclear what the exact cause of this is. It could be related to properties

associated with the particular training algorithm used, or it may be a result of overparameterization. Given the fact that an increased number of nodes per layer increases the training time of a FFN (4 nodes per layer takes less than a minute, 28 nodes takes 32 minutes), limiting our structure to 16 nodes produces the best results within reasonable amounts of computation time (about 13 minutes).

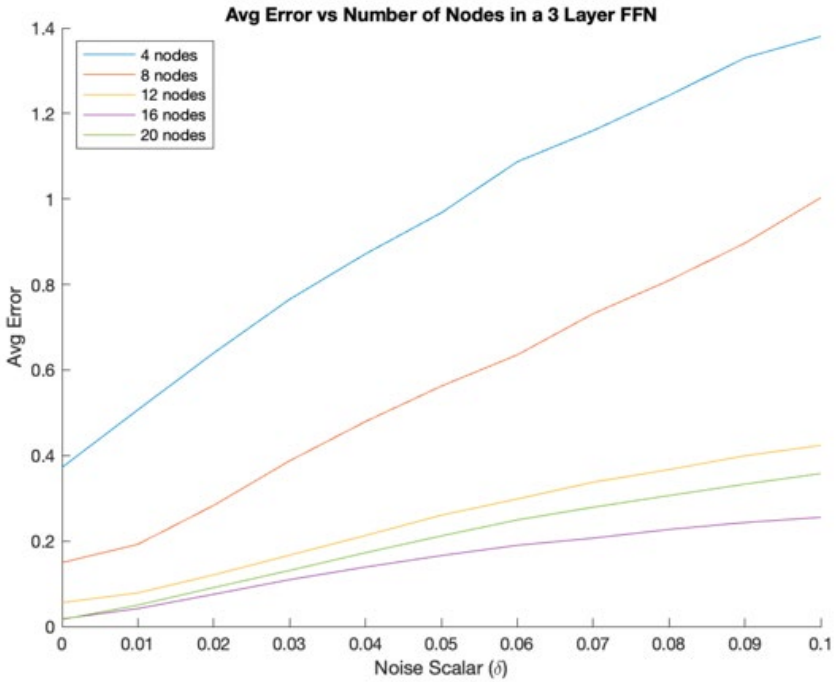


Figure 3.10. Error of FNN with 3-layers and varying number of nodes

2. Robustness under uniformly random noise

Based on the previous two tests of network structure, we proceeded with a 3-layer, 16 nodes per layer feedforward neural network to test the robustness. Note that this network was trained using "clean" data (i.e., no added noise), so each of the three altered data sets with noise presents a new challenge for the network. Various types of noise/perturbation are introduced in Section III.C.5. Figure 3.11 shows the results of this study, with the degree of noise (noise scalar) vs the average error of predictions. Figure 3.12 shows the error distributions for each type of noise with $\delta = 0.05$ and Table 3.1 provides the numerical values for the cumulative distribution.

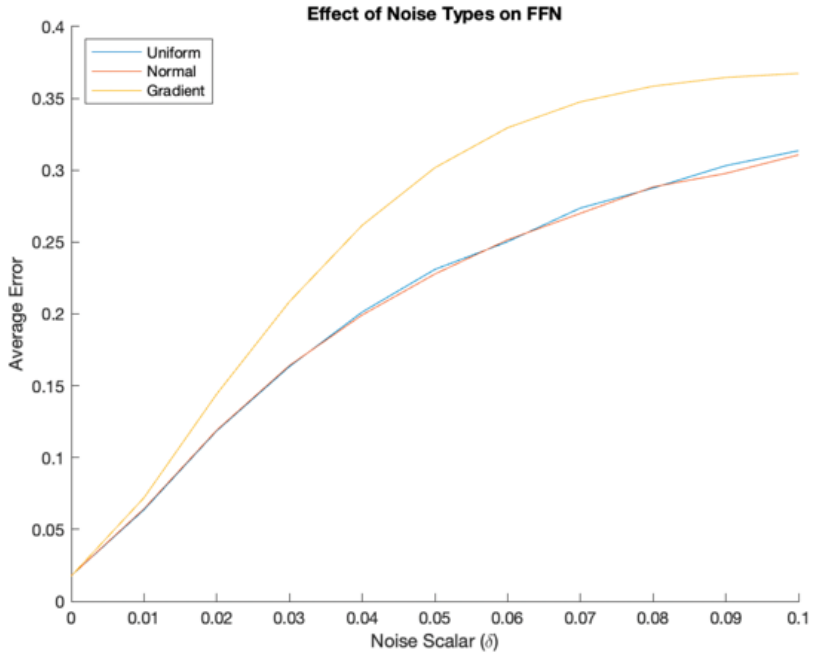


Figure 3.11. Effect of different noise types on a 3-layer, 16-node FFN.

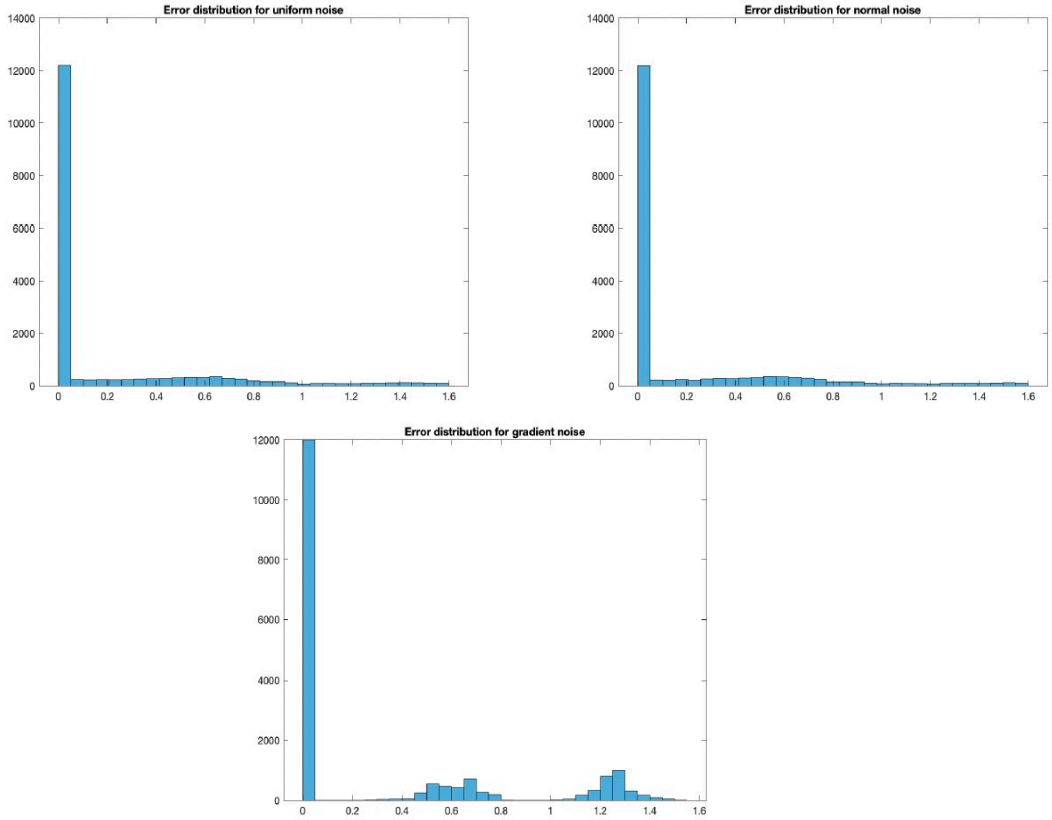


Figure 3.12. Error distribution for a 3-layer, 16 node FFN against uniform, normal, and gradient perturbations (δ = 0.05).

	< 0.5	< 1.0	< 1.5	< 2.0
Uniform	0.7984	0.9241	0.9749	1.0000
Normal	0.7998	0.9268	0.9743	1.0000
Gradient	0.6785	0.8339	0.9967	1.0000

Table 3.1. Percent of predictions within error bounds.

For small amounts of noise, the difference between types is not significant, however, as the noise scalar increases, we see a clear distinction between random noise (uniform or normal) and gradient perturbations. As expected, the gradient perturbations cause the highest error since their direction is chosen based on the greatest change in the function. For random noise, however, the distribution does not seem to matter provided that the mean ($\mu = 0$) and standard deviation ($\sigma = 1$) are the same. When looking at the error distributions, it is however clear that the majority of data points are classified correctly, regardless of the type of noise involved. The difference being that the gradient noise causes significantly more error in the $1.0 - 1.5$ range. While this does cause a higher average error, in practice the difference between an error of 0.75 and 1.25 is insignificant given the fact that both predictions would be misclassified after rounding. The most relevant metric for accurate predictions on this data set is the percent of predictions with an error < 0.5.

3. Network types and their robustness

For different types of networks, we test their robustness against data with added noise. The feedforward net (our baseline structure) is compared to the pattern net and cascade forward net, as described in section III.B. All three networks have the same structure with 3 layers and 16 nodes per layer. Here we use the percent of data points which are misclassified as the error measurement since the pattern net is a classification method rather than a regression tool. We round the predictions of the FFN and CFN so that all three networks output an integer prediction to represent the power grid fault location. We must also note that the PN does not lend itself to gradient perturbations, since the gradient cannot be easily computed with the classification nature of the network structure. The results for uniform and Gaussian noise are shown in Figures 3.13 and 3.14.

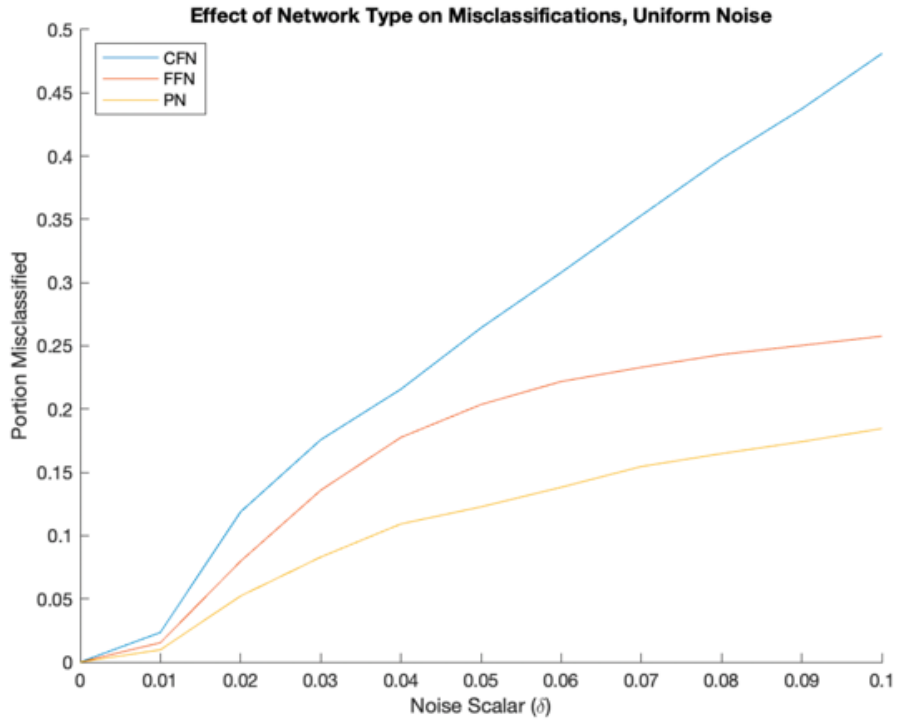


Figure 3.13. Comparison of network types against uniform noise

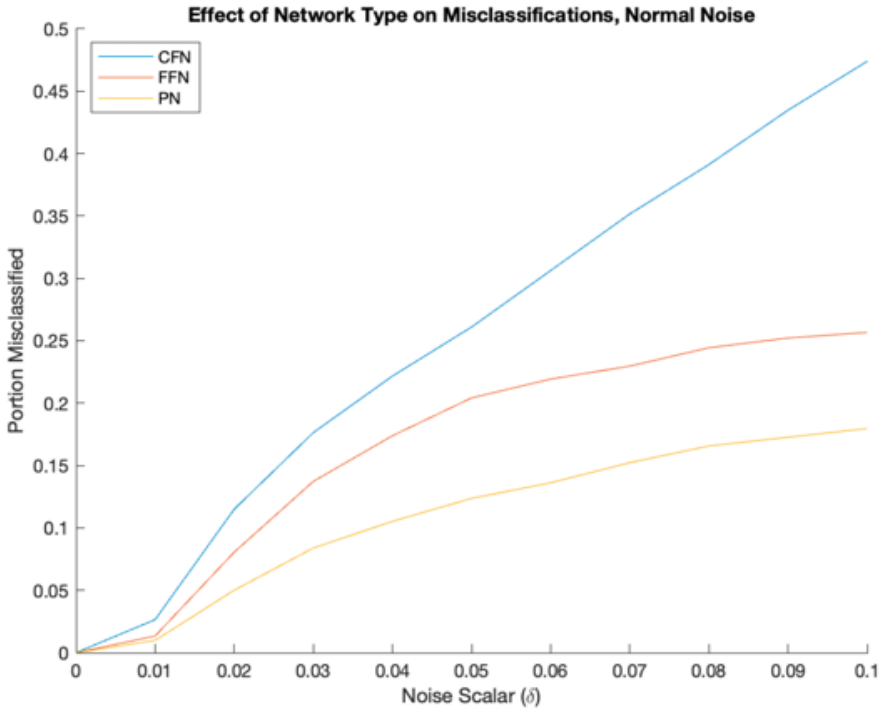


Figure 3.14. Comparison of network types against normal noise

The two types of random noise produce nearly identical graphs, again confirming that the distribution matters less than the mean and standard deviation. Here we clearly see that the pattern net produces the least misclassifications whereas the cascade forward net produces the most. This is likely due to the fact that the pattern net is designed for classification problems, whereas the other two are used for regression and then rounded. Between the FFN and CFN it is interesting to see that the CFN, with far more connections between nodes and layers, actually performs worse than the simple FFN. Considering the additional training time requirements for a CFN, the FFN is clearly a better choice. Finally, we see that the misclassification rate for a CFN follows a near linear relationship with the noise scalar, whereas the FFN and PN both start to level off as the noise scalar increases. Once again, this is an advantage of the FFN and PN.

4. The Stability of Prediction

For further comparison between networks, we graphed their stability of prediction (SOP) defined in Section III.A, Definition 3.2. Figure 3.15 shows the data obtained for random noise and Figure 3.16 shows gradient perturbations. Each graph contains the Stability of Prediction values for the 3-layer FFN, 3-layer CFN, 9-layer FFN, and 9-layer CFN (each with 16 nodes per layer). Note that the PN networks are not represented due to the inherent differences in their classification method as opposed to regression. Once again, we see virtually no difference between the two types of random noise, confirming the fact that the different distributions have the same effect so long as their mean and standard deviation are identical. On the other hand, when it comes to the gradient perturbations, a clearly different pattern emerges between the network types, particularly in the leveling off and then sharp downturn of the 3-layer CFN. All three graphs show that the 9-layer CFN is least stable with high noise levels, whereas the 9-layer FFN is most stable. Interestingly, the 3-layer CFN performs worse for smaller levels of noise and the 3-layer FFN performs slightly better. Regardless of the noise scalar, we see that the FFN structure is generally more stable than the CFN. Taken together, these graphs demonstrate how a grouping of networks could be used to determine whether noise has been added to the system. While all four networks perform with perfect accuracy on "clean" data, each network is affected differently by random or adversarial noise. These

differences in stability could be measured and compared to network-noise profiles to classify the type and scale of added noise.

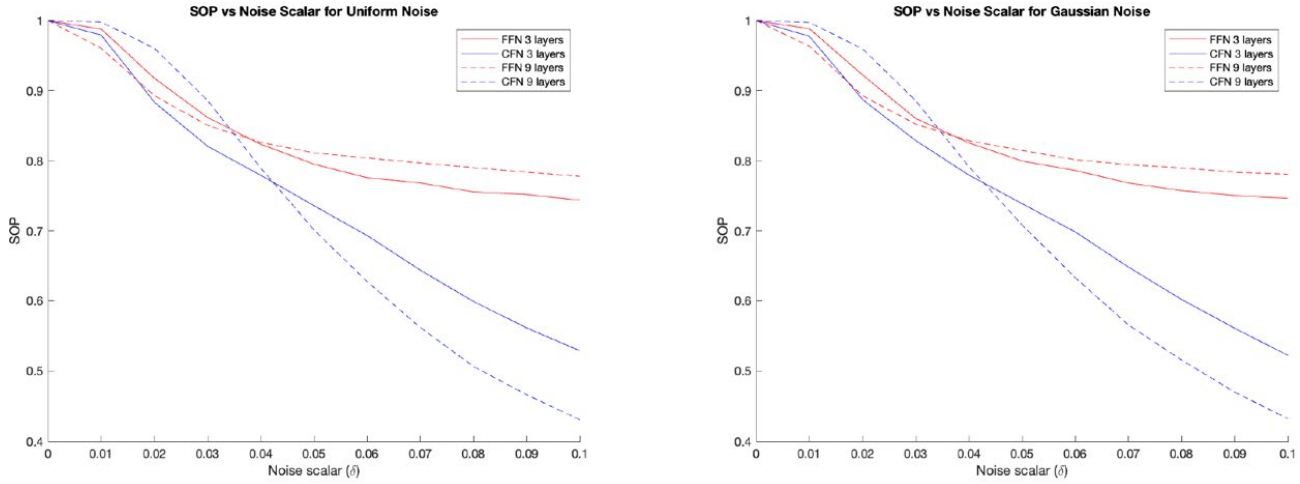


Figure 3.15. SOP comparison for 4 network structures with random added noise.

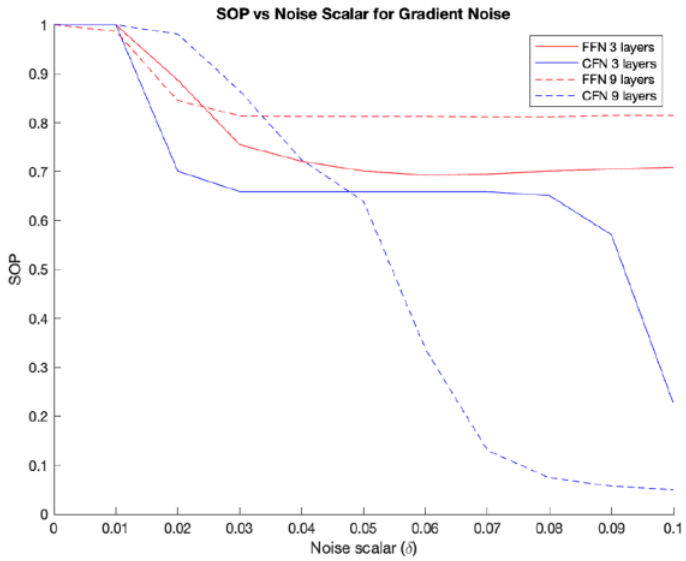


Figure 3.16. SOP comparison for 4 network structures with gradient perturbations.

5. Learning with noisy data

Since the prediction ability of a neural network is dependent on its training, we suggest using noisy training data to produce a more robust network. Using the same baseline FFN

structure with 3 layers and 16 nodes each, we trained new networks using "noisy" data sets. These were produced by adding uniform noise to the original training data set. We trained one network for each noise scalar (δ) and then tested it against the validation data with added uniform noise. The results are shown in Figure 3.17, where each line represents a network trained with added noise. The graph shows that the average error follows a curve with the lowest point at a testing noise scalar slightly lower than the training noise scalar. For example, the red line, representing a network trained with added noise of $\delta = 0.04$, produces the lowest error when the testing data has noise of $\delta = 0.03$. This trend is shown in all five networks, implying that one should train their network with slightly noisier data than what they expect to test it on. For this data set and range of testing noise scalars, the overall lowest error is from a network trained on added noise of $\delta = 0.08$ (the purple line).

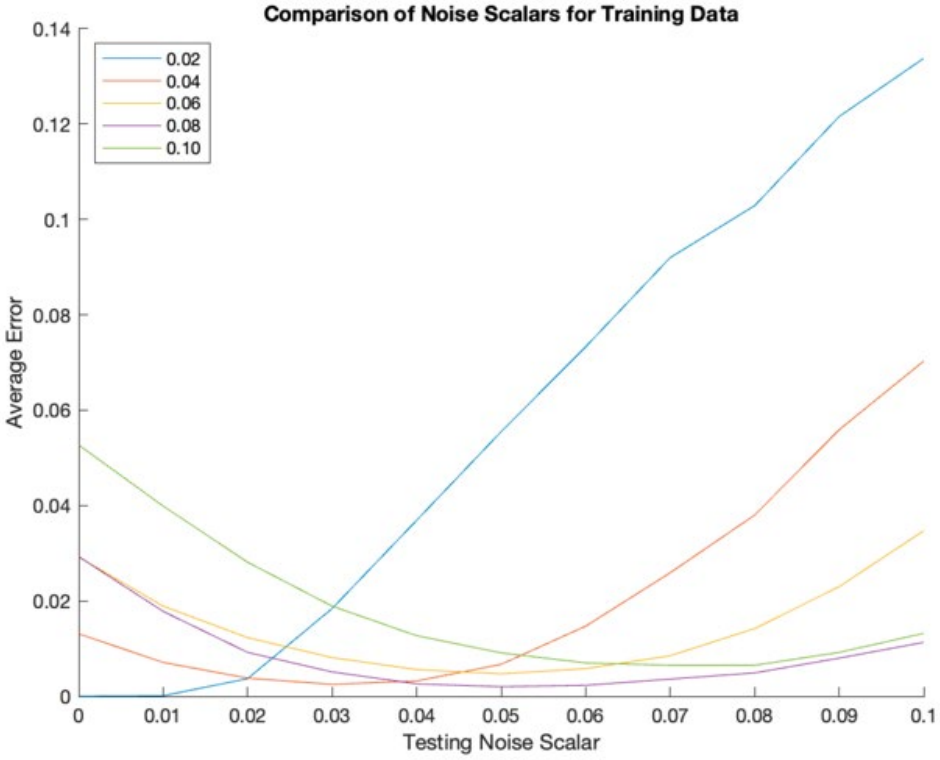


Figure 3.17. Effect of uniform noise added to both training and validation data

6. Learning using real data

Following the same procedure as Section III.D.1, we train several FFNs with structures that vary in the number of layers and nodes per layer. Figure 3.18 shows the average errors for networks with 16 nodes per layer and various numbers of layers, where each colored line represents testing data with a different noise scalar (δ). We see very little difference between the noise scalars; however, we see a similar pattern in the steep decline up to 3 layers, then a non-linear relationship emerges as the layers increase. While further study is certainly required for this real data set, we do see interesting comparisons with the simulated data results from Section III.D.1. For both, the optimal configuration appears to be 3 layers with 16 nodes per layer. However, in the case of real data, added noise does not appear to have the same adverse effect it had on the simulated data (Figure 3.19). This could be due to the already noisy nature of real data.

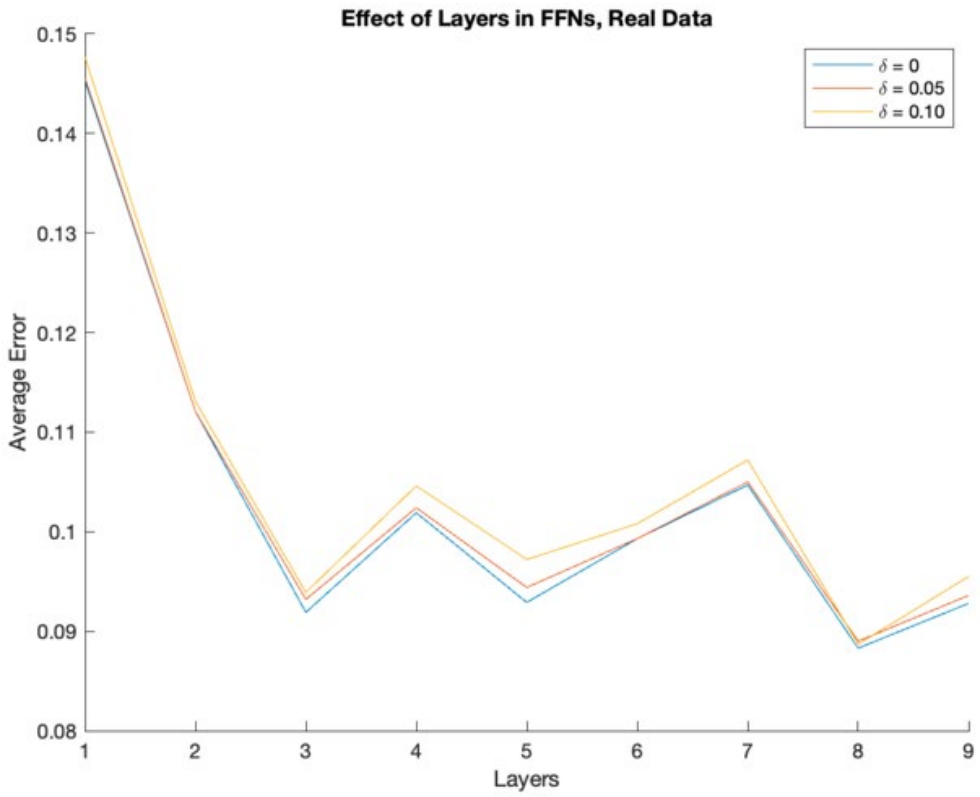


Figure 3.18. Effect of number of layers in a FFN, each with 16 nodes per layer.

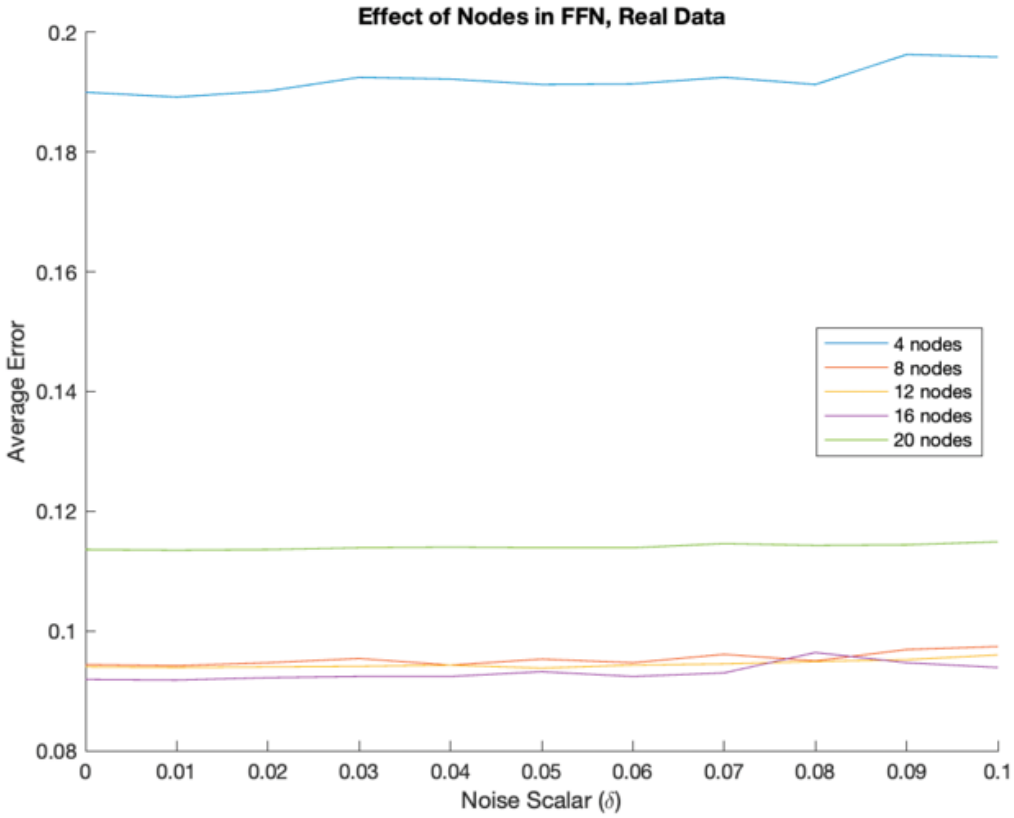


Figure 3.19. Effect of number of nodes per layer in a FFN, each with 3 layers

E. SUMMARY

We used MATLAB's Deep Neural Network Toolbox and trained networks to predict anomaly of dynamical systems, a task of infrastructure cyber security. The basic ideas of robustness analysis are illustrated using a microgrid model. These networks were challenged by adding both random noise, gradient noise, and adversarial noise to the testing data. We compared the performance of various network structures, analyzed each type of noise, graphed the Stability of Prediction for multiple types of networks, and improved robustness by adding noise to the training data. Using this type of analysis, one can determine neural network robustness. For the microgrid example, the analysis leads us to a robust network that has 3 layers and 16 nodes per layer. While this "ideal" structure is specific to the 9-bus microgrid data, we can assume that other data sets will also have an "ideal" network structure. Interestingly, more layers and more nodes does not always produce a more accurate neural network. Additionally, deeper networks

require more time to train, so achieving results with a simpler network can significantly reduce the overall time to build an effective machine learning system. Overall, we conclude that adding layers and nodes to a feedforward neural network does not directly improve its robustness. Adding random noise from both the uniform and normal distributions showed no difference in their effects on the network's prediction ability. We hypothesized that different noise distribution might affect the predictions, but this did not appear to be the case. Instead, we conclude that noise drawn from any distribution with the same mean and standard deviation will produce the same or very similar results in the network. Adding adversarial noise in the direction of the network gradient did have a significantly different effect from the random noise distributions. However, the computation shows that more complex networks (i.e., the CFN with more connections between layers) does not necessarily result in a more robust network. This phenomenon is consistent with our first conclusion based on random noise. Additionally, we observe that the PN, which is identical in structure to the FFN, produces better results because it is designed for classification rather than regression. By plotting the Stability of Prediction for several networks against added noise of each type, we see patterns emerge that may be useful for noise classification. As expected, the uniform and normal random noise graphs are nearly identical, but the gradient noise has a distinctly different shape. Finally, we added uniform random noise to the training data as a way in which to improve the robustness of our networks against noisy testing data. Testing shows that the lowest error rates were achieved when a network was trained with slightly higher noise levels than those present in the testing data. This conclusion is important for real-world applications, where noise is expected in the input data and thus the ML system should be trained with additional noise added to the training data set. Overall, we found that the FNN achieved a very good level of robustness provided that optimal hyperparameters can be found. In the next step, we will study the interconnection between the dynamical behavior of the problem and the robustness, trying to reveal the fundamental reason of the robust performance that is found in the microgrid example. In addition, we will continue to analyze the learning-based method and its robustness based on the real data collected from the FDR sensors.

IV. DISTRIBUTIONAL ROBUSTNESS

It is pointed out in Section II.D that 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 study, we 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.

A. DATA STRUCTURE AND DATA GENERATION

1. Data structure

Our data was simulated as a traditional IEEE 9-bus model introduced in III.C.1. It has three power generators, numbered 1, 2, and 3. 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. 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 the case of no fault was represented by the number 0. This matrix was

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

where $k_i \in \{0, 1, 2, 3\}$.

2. Simulated Data

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 above. We studied a random one-second section of each trajectory, taking measurements from all three generators at 10 Hz, giving us a 30 × 20,000 input data set on which to train. Our predictions would attempt to match a 1 × 20,000 vector of fault numbers. To find the optimal network size, we trained and tested neural networks with one to five layers, each containing between 1 and 16 nodes per layer, using the hyperbolic tangent, sigmoid, and rectified linear activation functions. In this research, we focused 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.

We tested the neural networks 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. 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]. Our normal distribution and log-normal distribution are linearly transformed into the domain of [−0.05, 0.05]. Generating a 30 × 20,000 testing set for each of these, our initial neural network performed perfectly. 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 $\sigma = 0$ and $\sigma = \sqrt{1/1200}$., and the log-normal distribution used $\mu = \log(0.05) - \log(4/3)/2$ and $\sigma = \log(4/3)$, linearly transformed into the domain of [−0.05, 0.05]. 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.

B. RESULTS AND DYNAMICS ANALYSIS

One important question was how we could determine 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 angles at $t = 0$ would follow the given distribution around the equilibrium point of the rotor, but the rotor angles of data points randomly selected along

the system's trajectories 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.

1. Network Construction

Our first task was to optimize our neural network's parameters and hyperparameters to achieve the best possible predictions. The two performance metrics used were 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. 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.

		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.

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, visualized in Figure 4.1, 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 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.

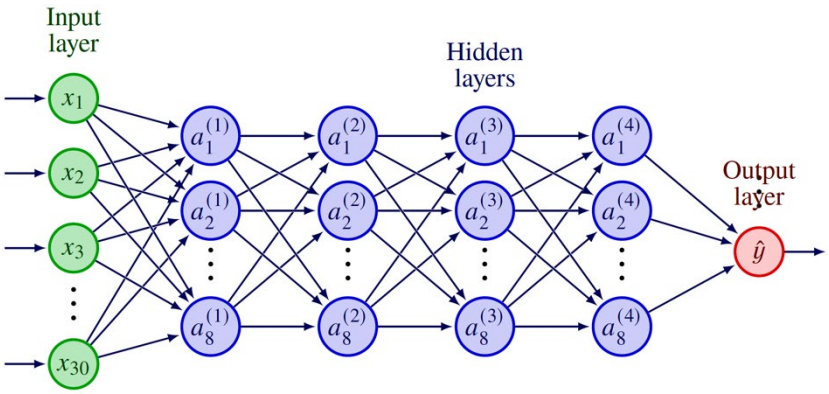
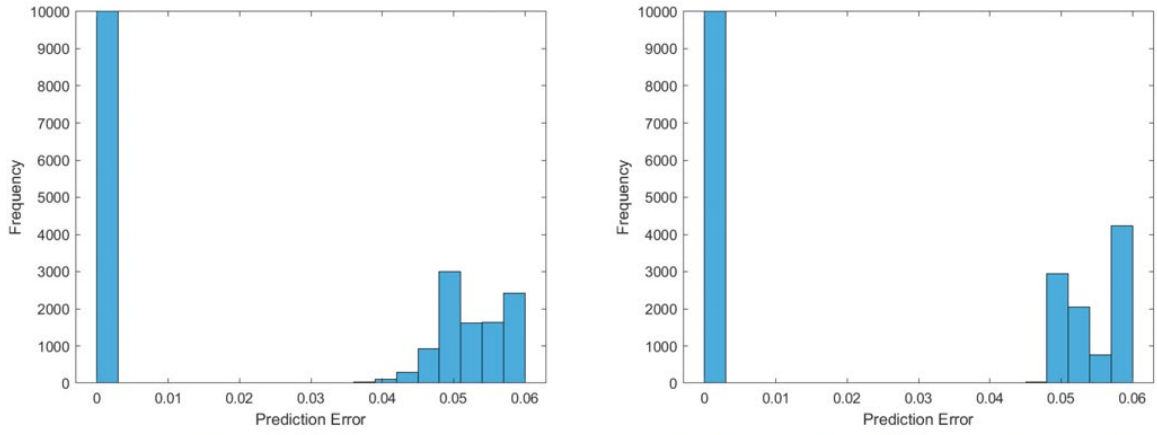


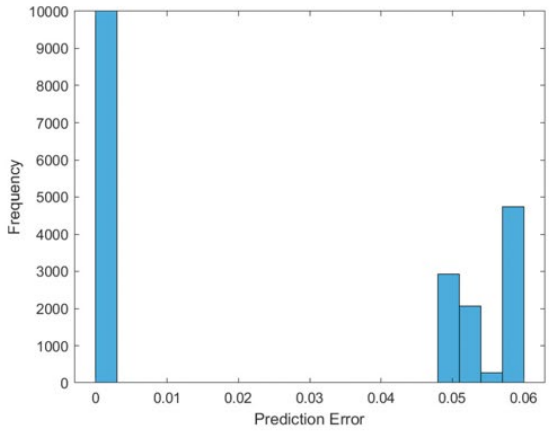
Figure 4.1. A visualization of our first optimum neural network, with four hidden layers and eight nodes per layer. Bias terms have been omitted for clarity.

A 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.



(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 vali- dation 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 stability of the dynamical system, i.e., the trajectory of the power system converges to similar curves that are independent of the initial state distribution.

We decided to change both our distributions and our network hyperparameters to try and explain this accuracy. First, we chose to 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. Subsequently, we applied 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.

2. Distributional changes

We tested the neural network depicted 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 our uniform distribution. This normal distribution had the same mean and standard deviation as the original training uniform distribution, with $\mu = 0$ and $\sigma = \sqrt{1/1200} \approx 0.0289$. Using our original network composed of four hidden layers of eight nodes each, our RMSE was 0.119 and our misclassification rate was 0.

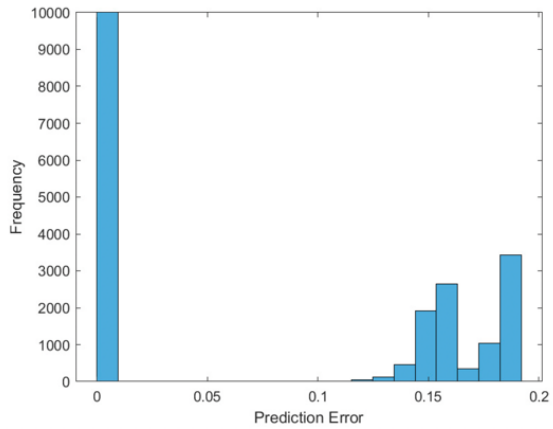
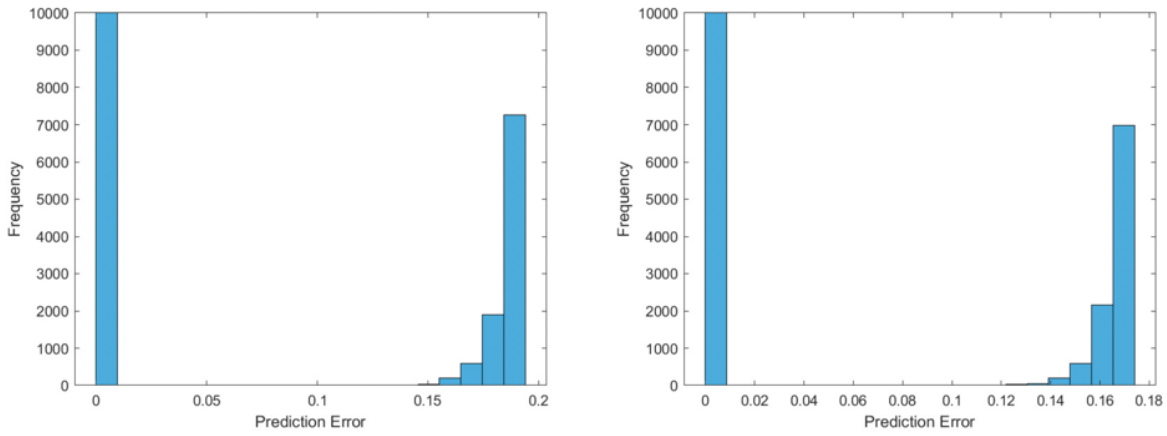


Figure 4.3. Prediction errors of a neural network composed of four hidden layers of eight nodes each on normally distributed data with $\mu = 0$ and $\sigma = \sqrt{1/1200} \approx 0.0289$.

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 $\sigma = \sqrt{1/1200}$. To create a log-normal distribution X, we considered a normal distribution $Y = \log(X)$, with $\mu_Y = \log(0.05) - \log(4/3)/2$ and $\sigma_Y = \sqrt{\log(4/3)}$. This created a single, right skew log-normal distribution X with a standard deviation that matches the original uniform distribution, but with $\mu = 0.05$.

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 distribution, our RMSE was 0.117, and on our left-skew 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.



(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.

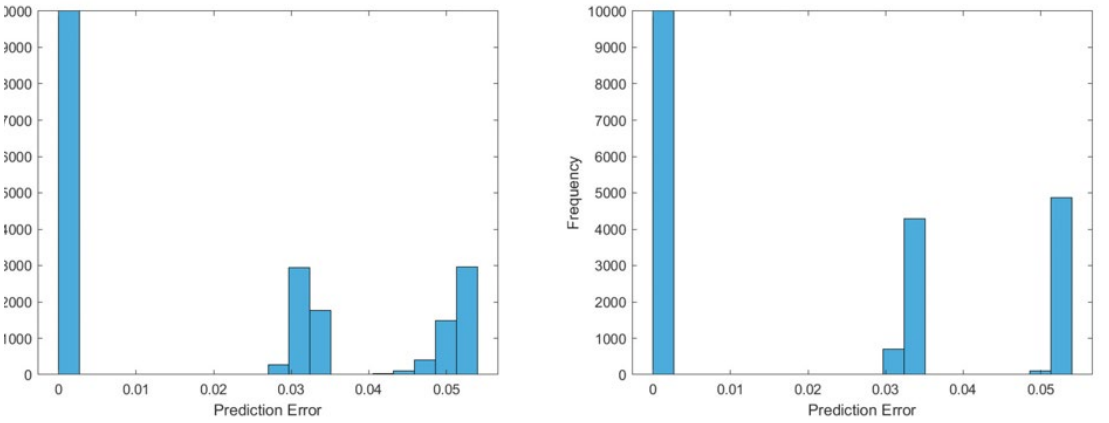
To summarize, widening the standard deviation made our predictions slightly less accurate, but a similar error pattern is still clearly visible in the histogram in Figure 4.3. While the error groupings were somewhat condensed in Figure 4.4, the pattern of a section of near- perfect predictions followed 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.

3. Depth and width changes

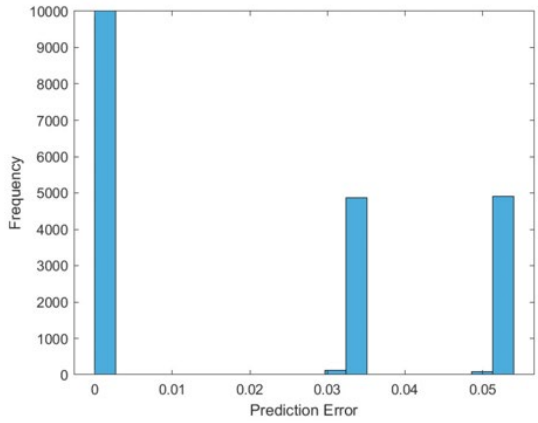
Our next consideration was to examine the influence of the depth or width of the neural network on distributional robustness. A network's depth is equal to the total number of layers in the network, except for the input and output layers, and the width of a layer is the number of nodes in that layer. In this section, we will use the normal and log-normal distributions.

(i) 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, although they 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. 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.



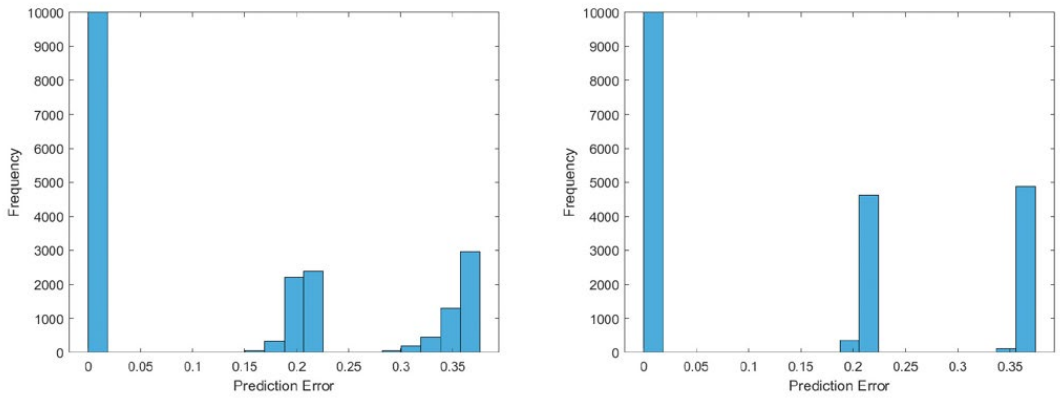
(a) *Uniform distribution* **(b)** *Normal distribution*



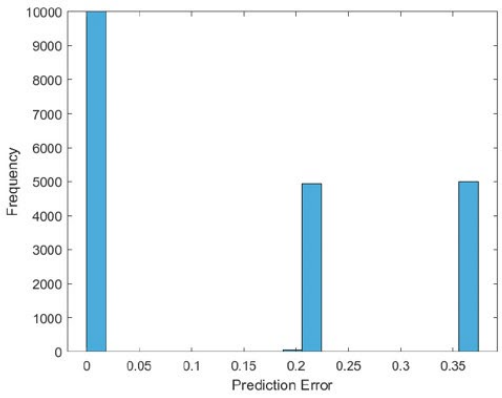
(c) *Log-normal distribution*

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.

This adjustment to depth did however 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 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.



(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.

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 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.

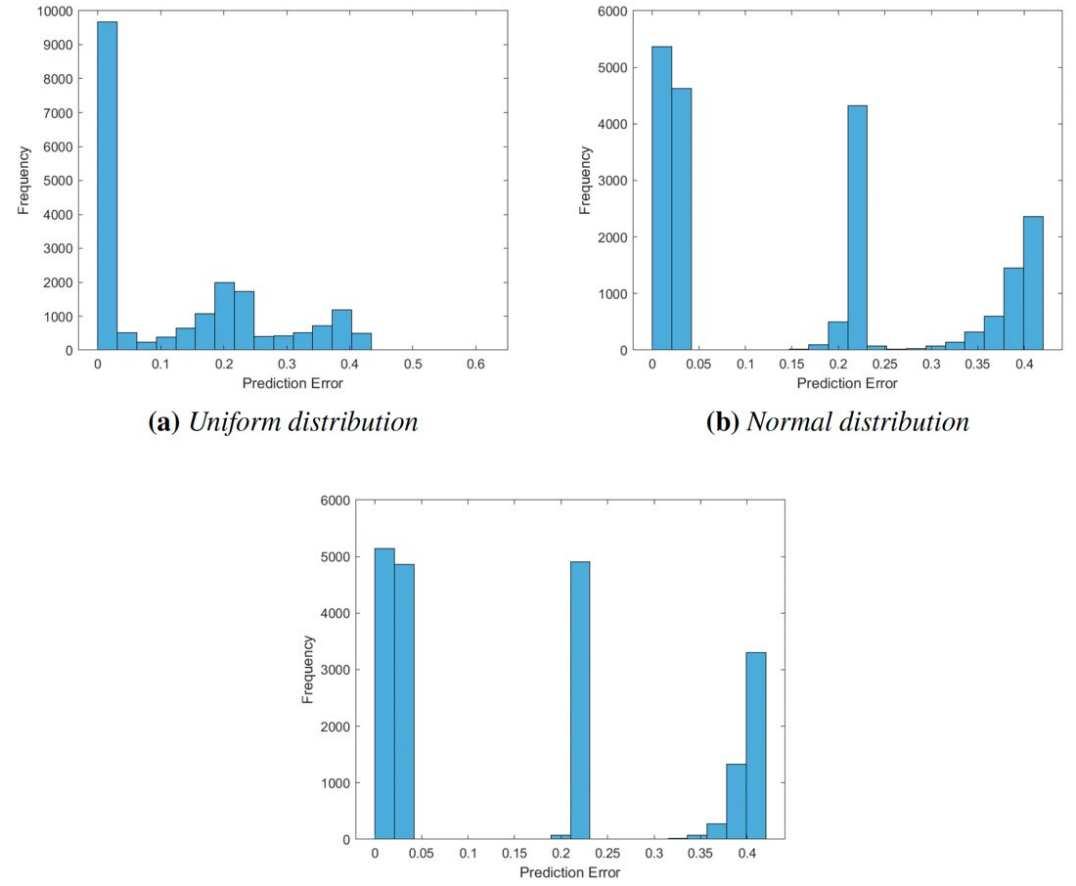


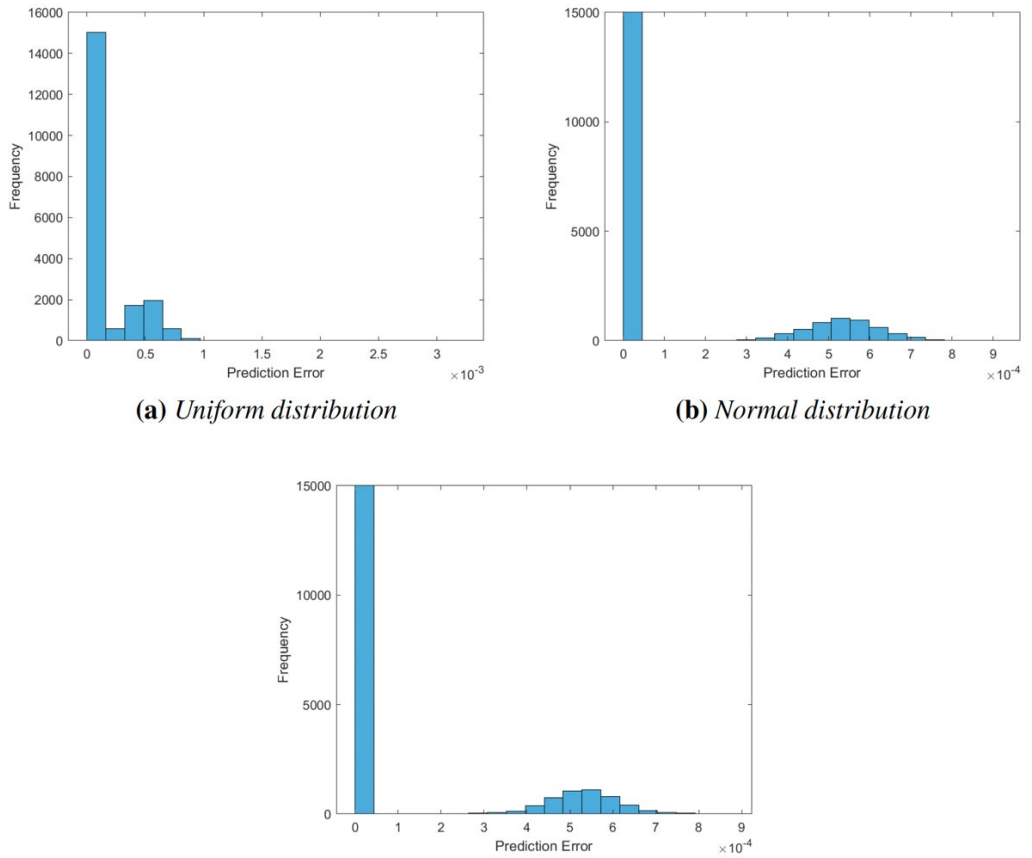
Figure 4.7. Error histograms of a network with a single hidden layer of eight nodes each on validation data with uniformly, normally, and log-normally distributed initial condition variation.

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, one can 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.

(ii) 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 outperformed our original base network consisting of four layers each with eight nodes, this trend should not continue when looking at narrower neural networks.



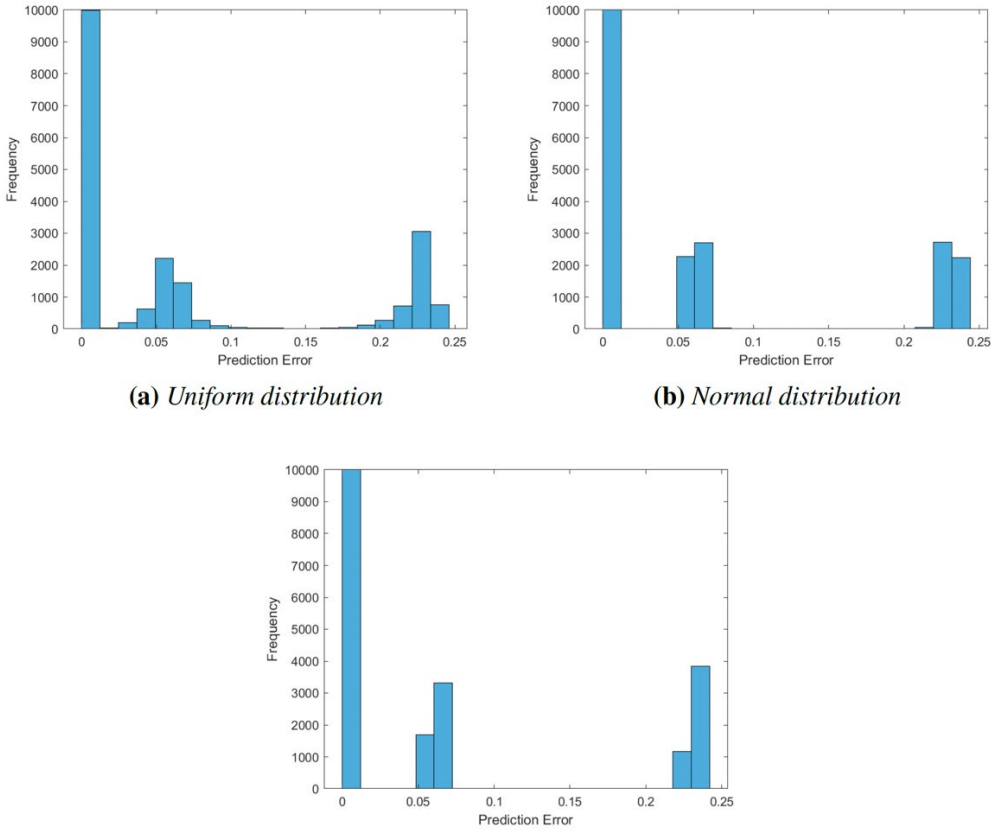
(a) *Uniform distribution*

(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.

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.



(a) *Uniform distribution*

(b) *Normal distribution*

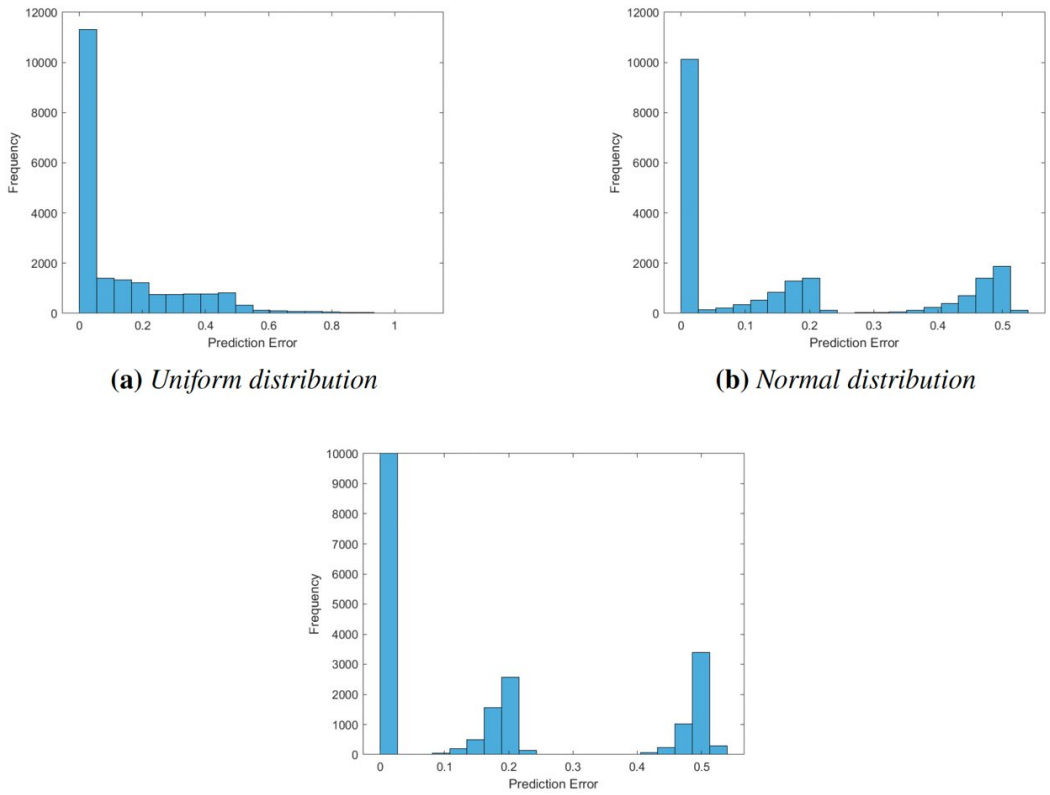
(c) *Log-normal distribution*

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.

(iii) 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. 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 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.



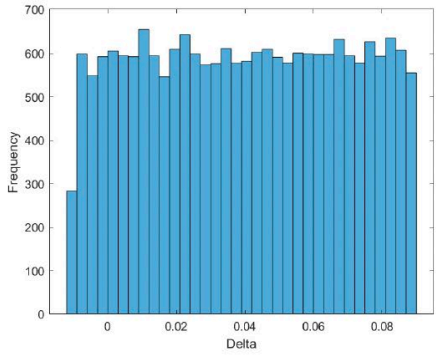
(a) *Uniform distribution* **(b)** *Normal distribution*

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.

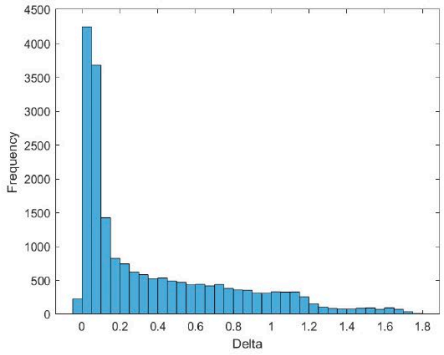
4. Data distribution of dynamic systems

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 level of robustness and determine 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. Our conjecture being that the high level of robustness is a result of the fact that 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

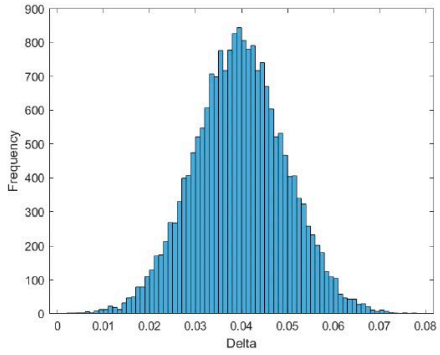
when we sample our one-second window of data. To see what this would look like, we analyzed our initial condition variation for the first generator δ_1 at both the initial time t = 0, which would have a uniform, normal, or log-normal distribution, and at the beginning of the random one-second window. Results are included in Figure 4.11.
At t = 0, our initial condition for all three distributions was predictably centered around the equilibrium for δ_1, which was 0.0396. However, the distribution of our data points at random time along trajectories was almost identical, with a large number of samples at the equilibrium itself and the rest following a similar pattern across all data distributions. In other words, no matter what the initial state distribution was, the data set distribution for training and validation converges due to the laws of dynamics in our system. Changing the distribution did however 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.



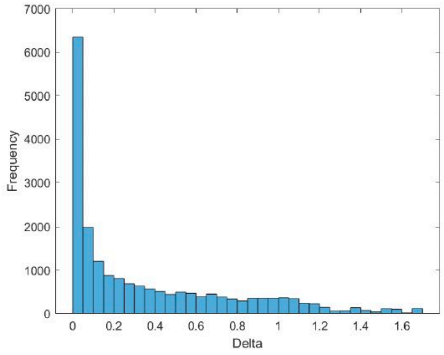
(a) *Uniform distribution at* $t = 0$



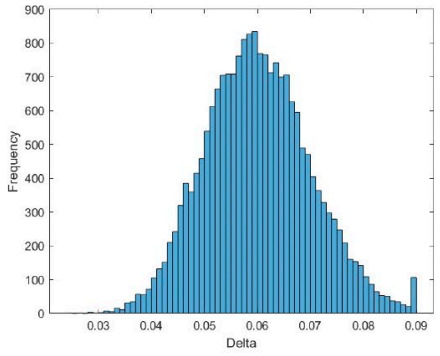
(b) *Uniform distribution at random time*



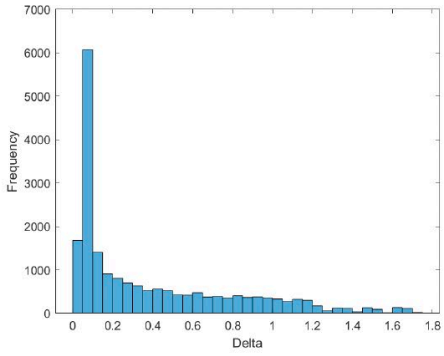
(c) *Normal distribution at* $t = 0$



(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 c1 choices at t = 0 and at a randomly selected time, with uniform, normal, and log-normal distributions.

C. SUMMARY

This study aimed to introduce a new form of robustness for neural network users to consider in the form of distributional robustness for dynamical systems. 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. 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 IV.B.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 IV.B.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 errors. 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 and must be considered when applying neural networks in real-world scenarios.

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 in 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 the distributions of our data points at random one-second time window along trajectories to converge to a similar pattern, regardless of the initial state distributions. If we were applying a neural network to a real-world microgrid or other dynamics-dominated scenarios, 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 study 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 study 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 general machine learning performance.

Future study of machine learning robustness may focus on real-world 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 IV.B.2 and IV.B.3 are products of our microgrid application, choice of feedforward neural networks as a ML system, or a feature visible in other studies of distributional robustness.

V. CONCLUSIONS

The goal of this project was to investigate mathematical concepts and quantitative measures of robustness and vulnerability to adversarial data for cybersecurity DL applications, as well as develop computational methods capable of quantitatively evaluating the robustness and vulnerability of DL tools. The first phase of the project was a literature review, which was covered in Section II of the report. During this phase of our research, we discovered that the concept of robustness is not unique, or sometimes even rigorously defined, in the literature. There are a variety of different aspects of robustness that may affect the performance of a ML model, and several papers we reviewed highlighted the fact that improving robustness oftentimes can adversely impact the performance of the specific model in use. The study and evaluation of robustness should therefore be carried out with the application in mind. Research involving machine learning in network - and cybersecurity settings is less prevalent than other application areas. Most studies on ML robustness have thus far focused on applications such as computer vision, image classification, language processing, etc. Some techniques in the existing literature are applicable to a wide spectrum of application scenarios. However, the robustness of machine learning employed in cybersecurity, computer networks, and infrastructure settings is an important area that calls for more research.

The second phase of the project, reported in Section II, was focused on the robustness analysis of infrastructure cybersecurity. Using a microgrid power system model and learning-based fault detection as the testbed, we investigated the robustness of neural networks under noisy or poisoned data. We found that noise drawn from a variety of different distributions with the same mean and standard deviation produce the same, or very similar, result in the neural network when detecting faults. However, our research revealed that adding adversarial noise in the direction of the network gradient did have a significantly different effect from the random noise distributions. In addition, our investigation showed that more complex networks (i.e., the CFN with more connections between layers) did not necessarily result in a more robust network. To improve the robustness of our networks when operating in noisy environments, we also added

uniform random noise to the training data. This approach proved to be very successful. Our lowest error rates were achieved when our networks were trained with slightly higher noise levels than those present in the testing data. This result is important for real-world applications. In cases where natural or adversarial noise is expected in the input data, we recommend training the ML model with additional noise present in the training data set.

The third phase of the project was focused on investigating distributional robustness. Neural networks may sometimes be used outside of the environment in which they were trained. If the incoming data's distribution is significantly different from that of the training data, it may negatively impact the performance of the ML system to a greater or lesser degree. Quantifying this degree and measuring the robustness of the network is essential. Using fault detection of a microgrid as an example, we were able to see the impact of changing distributions on neural network performance. As our initial condition variation for our generator rotor angular position moved further from the variation we used for training, we observed a slight decrease in ML performance. However, the misclassification rates remained low, indicating that our neural network in general remained robust. In this study, neural networks trained for power system's fault detection exhibited good overall robustness. We believe the primary reason for this stems from the power system's dynamical behavior; a phenomenon that is significantly different from that experienced in other applications such as image processing. By selecting random pieces of data from dynamic trajectories of the power system, one is more likely to obtain similar looking data samples, since the system is given a chance to approach a steady state. While the initial distributions for trajectories of the differential equations can be vastly different, the dynamic nature of our microgrid causes the distributions of our data points along trajectories to converge to a similar pattern, regardless of the initial state distributions. When applying a neural network to a real-world dynamics-dominated scenario, this could be advantageous, since the network could operate accurately regardless of distributional changes.

LIST OF REFERENCES

- P. M. Anderson and A. A. Fouad (2008), Power system control and stability, John Wiley & Sons.
- I. Almutairy and M. Alluhaidan (2017), "Fault diagnosis based approach to protecting dc microgrid using machine learning technique," Procedia Computer Science, vol. 114, pp. 449–456.
- K. Alrawashdeh and C. Purdy (2016), Toward an online anomaly intrusion detection system based on deep learning, 15th IEEE International Conference on Machine Learning and Applications, Doi 10/1109/ICMLA/2016.167.
- E. Anthi, L. Williams, M. Rhode, P. Burnap, and A. Wedgbury (2021), "Adversarial attacks on machine learning cybersecurity defenses in industrial control systems," Journal of Information Security and Applications, vol. 58, p. 102717. Available: https://www.sciencedirect.com/science/article/pii/S2214212620308607.
- M. Barreno, B. Nelson, A. Joseph, and J. Tygar (2010), "The security of machine learning," Machine Learning, vol. 81, pp. 121–148.
- D. S. Berman, A. L. Buczak, J. S. Chavis and C. L. Corbett (2019), A survey of deep learning methods for cyber security, Information, 10(4), 122; doi:10.3390/info10040122.
- N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal (2018) "Enhancing robustness of machine learning systems via data transformations," in 2018 52nd Annual Conference on Information Sciences and Systems (CISS). IEEE, pp. 1–5.
- T. Caelli, W. F. Bischof, and W. F. Bischof (1997), Machine Learning and Image Interpretation. New York, NY, USA: Springer Science & Business Media.
- Connecticut Department of Energy and Environmental Protection (2020), "Microgrid grand and loan programs," Jan. 2020 [Online]. Available: https://portal.ct.gov/DEEP/Energy/Microgrid-Grant-and-Loan/Microgrid-Grant-and-Loan-Program

- E. Derks, M. S. Pastor, and L. Buydens (1995), "Robustness analysis of radial base function and multi-layered feed-forward neural network models," Chemometrics and Intelligent Laboratory Systems, vol. 28, no. 1, pp. 49–60.
- GAO@100 (2021), Report to Congressional Requests: Electricity grid cyber security – DOE needs to ensure its plans fully address risks to distribution systems, United States Government Accountability Office, https://www.gao.gov/products/gao-21-81.
- R. Gao and A. J. Kleywegt (2016), Distributionally robust stochastic optimization with Wasserstein distance, arXiv:1604.02199.
- T. K. Hembram, S. Saha, B. Pradhan, K. N. Abdul Maulud, and A. M. Alamri (2021), "Robustness analysis of machine learning classifiers in predicting spatial gully erosion susceptibility with altered training samples," Geomatics, Natural Hazards and Risk, vol. 12, no. 1, pp. 794–828.
- D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song (2019), "Using self-supervised learning can improve model robustness and uncertainty," Advances in Neural Information Processing Systems, vol. 32, 2019.
- R. C. B. Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan (2014), "Machine learning for power system disturbance and cyber-attack discrimination," in the 7th IEEE International symposium on resilient control systems (ISRCS), pp. 1–8.
- H. Lin, K. Sun, Z.-H. Tan, C. Liu, J. M. Guerrero, and J. C. Vasquez (2019), "Adaptive protection combined with machine learning for microgrids," IET Generation, Transmission & Distribution, vol. 13, no. 6, pp. 770–779, 2019.
- O. Linda, T. Vollmer, and M. Manic (2009), "Neural network-based intrusion detection system for critical infrastructures," Proceedings of the International Joint Conference on Neural Networks, pp. 1827–1834, June 2009.
- N. Lu, S. Mabu, T. Wang, and K. Hirasawa (2013), "An efficient class association rule-pruning method for unified intrusion detection system using genetic algorithm," IEEJ Transactions on Electrical and Electronic Engineering, vol. 8, March 2013.

- MathWorks Help Center (2022), "trainlm Levenberg-Marquardt backpropagation," accessed May 10, 2022 [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/trainlm.html.
- A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang, and S. Tandon (2017), UFLDL tutorial, [Online]. Available: http://ufldl.stanford.edu/tutorial/.
- M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre (2004), "Collaborative recommendation: A robustness analysis," ACM Transactions on Internet Technology (TOIT), vol. 4, no. 4, pp. 344–377.
- S. Rahman Fahim, S. K. Sarker, S. M. Muyeen, M. R. I. Sheikh, and S. K. Das (2020), "Microgrid fault detection and classification: Machine learning based approach, comparison, and reviews," Energies, vol. 13, no. 13, 2020. Available: https:// www.mdpi. com/1996-1073/13/13/3460.
- A. Robey, H. Hassani, and G. J. Pappas (2020), "Model-based robust deep learning," arXiv preprint arXiv:2005.10247.
- D. E. Rumelhart, G. E. Hinton, R. J. Williams (1986), Learning representations by back-propagating errors. Nature. 323 (6088): 533–536, 1986.
- T. D. Sanger (1989), "Optimal unsupervised learning in a single-layer linear feedforward neural network," Neural networks, vol. 2, no. 6, pp. 459–473, 1989.
- I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters and A. Ng (2020), Cybersecurity data science: an overview from machine learning perspective, Journal of Big Data, 7(41), https://doi.org/10.1186/s40537-020-00318-5.
- S. S. Vallender (1974), Calculation of the Wasserstein distance between probability distributions on the line, Theory of Probability and Its Applications, 18(4), pp. 784-786.
- V. Vittal, J. D. McCalley, P. M. Anderson, and A. Fouad (2019), Power System Control and Stability. Hoboken, NJ, USA: John Wiley & Sons, 2019.
- University of Tennessee Knoxville document, "Fnet/grideye web display," accessed May 20, 2022 [Online]. Available: http://fnetpublic.utk.edu

- D. Wang, X. Wang, Y. Zhang, and L. Jin (2019), "Detection of power grid disturbances and cyber-attacks based on machine learning," Journal of information security and applications, vol. 46, pp. 42–52.
- E. Wood (2018), "Microgrid project at submarine base receives $5m Connecticut grant," Microgrid Knowledge, Sept. 10, 2018 [Online]. Available: https://microgridknowledge.com/microgrid-project-submarine-base/.
- W. Yu and J. Cao (2006), "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.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Research Sponsored Programs Office, Code 41
   Naval Postgraduate School
   Monterey, CA 93943

4. Navy Cyber Defense Operations Command
   112 Lake View Parkway
   Suffolk, VA 23435