2022-06

ROBUSTNESS OF MACHINE LEARNING FOR POWER SYSTEMS

Kozak, Elana P.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ROBUSTNESS OF MACHINE LEARNING FOR POWER SYSTEMS

by

Elana P. Kozak

June 2022

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

THIS PAGE INTENTIONALLY LEFT BLANK

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

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

4. TITLE AND SUBTITLE ROBUSTNESS OF MACHINE LEARNING FOR POWER SYSTEMS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Elana P. Kozak	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

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

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

13. ABSTRACT (maximum 200 words)

The applications of machine learning are broad and useful for a variety of industry and military objectives, but are the current methods robust? Robustness requires more than accuracy in ideal conditions; it means the system is resistant to perturbations in the data, both from natural and adversarial causes. This research aims to analyze the robustness of neural networks used for power-grid fault classifications. We focus on data generated from simulations of the classical 9-bus model; however, these methods and results can be extended to more complex microgrids, such as those found on naval ships, submarines, and bases. First, we measure the effects of random and adversarial noise on the testing data and compare three network types. Then we test different structures by varying the number of nodes and layers. Finally, we test whether adding noise to the training data can improve robustness. Before machine learning methods are adopted on submarines, we must first understand their weaknesses and potential for error. This research provides the foundation for how to test robustness, where neural networks are at risk from random or adversarial noise, and how to modify networks to improve their robustness.

14. SUBJECT TERMS machine learning, robustness, neural network, power grid	**15. NUMBER OF PAGES** 63
	16. PRICE CODE

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

NSN 7540-01-280-5500

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

ROBUSTNESS OF MACHINE LEARNING FOR POWER SYSTEMS

Elana P. Kozak
Ensign, United States Navy
BS, United States Naval Academy, 2021

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2022

Approved by: Wei Kang
Advisor

Thor Martinsen
Co-Advisor

Anthony J. Gannon
Second Reader

Francis X. Giraldo
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The applications of machine learning are broad and useful for a variety of industry and military objectives, but are the current methods robust? Robustness requires more than accuracy in ideal conditions; it means the system is resistant to perturbations in the data, both from natural and adversarial causes. This research aims to analyze the robustness of neural networks used for power-grid fault classifications. We focus on data generated from simulations of the classical 9-bus model; however, these methods and results can be extended to more complex microgrids, such as those found on naval ships, submarines, and bases. First, we measure the effects of random and adversarial noise on the testing data and compare three network types. Then we test different structures by varying the number of nodes and layers. Finally, we test whether adding noise to the training data can improve robustness. Before machine learning methods are adopted on submarines, we must first understand their weaknesses and potential for error. This research provides the foundation for how to test robustness, where neural networks are at risk from random or adversarial noise, and how to modify networks to improve their robustness.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

CFN cascade forward network

DNN deep neural network

FFN feedforward network

ML machine learning

MLF multi-layered feedforward

PN pattern network

RMSE root mean squared error

SOP stability of prediction

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank my advisors, Professor Wei Kang and CDR Thor Martinsen, for their support and guidance throughout this project. Their expertise and mentorship helped me grow in both my academic and professional life, making my time at the Naval Postgraduate School more valuable.

I would also like to thank Professors Anthony Gannon, Frank Giraldo, and Tron Omland for their input and suggestions on this work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Motivation

Machine learning (ML) is becoming a useful tool in many areas of life, including military operations. The ability for machines to rapidly assess large data sets is unparalleled to that of human analysts. Computational algorithms have provided us with many new ways of collecting information, solving problems, and drawing conclusions. Unfortunately, the decisions of a machine learning program are not always explainable or reliable. The inner workings of a neural network may not be transparent from its code, and most machine learning tools are vulnerable to errors, either from attacks or corrupted data. To harness all of the benefits from machine learning, we must first work to build trust in their functions. This requires a strong understanding of how the algorithms work and the limits to their reliability and robustness.

One area for machine learning to improve is power grid analysis. Many devices essential to our modern world are attached to a power grid, constantly drawing energy based on the device's operations. This energy usage can be measured, along with other data related to the power grid, as a function of time. Using machine learning techniques, we can draw conclusions about device operations based on data collected from sensors in the grid.

Microgrids are local energy grids that can be disconnected from a larger grid and operate independently [1]. These can be used in conjunction with a larger, traditional power grid to provide stability for critical infrastructure, such as Connecticut's Microgrid Program aims to do [1]. They can also be designed to operate completely independently, or "off the grid." The military already sees the value of microgrids, both for shore commands and vessels at sea. A nuclear submarine, of course, operates on its own microgrid, using electrical power generated by the reactor. As part of the Connecticut Microgrid Program, the U.S. Naval Submarine Base in Groton, CT, is also working on building a microgrid to power its submarines while in port [2]. Other military systems may be soon to follow, since microgrids rely less on the traditional power grid system and are more resilient and

adaptable to changing conditions.

For machine learning, microgrids provide a prime environment to analyze the system and make predictions off of 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 occurred, but this task is suited for machine learning programs as well [3]. Microgrids are particularly useful for studying these machine learning techniques because strategically measured data can sufficiently represent the workings of the power system, thereby allowing the neural network to make accurate predictions.

Despite the many applications and exciting potential of machine learning, there are weaknesses that can be exploited. Neural networks are only as good as the data they're trained on, and understanding the limitations of an ML system is crucial to implementing it. For military applications in particular, we must be absolutely confident in the system's ability to perform correctly under adverse conditions caused either from natural occurrences or deliberate attack. A submarine at sea cannot rely on a prediction algorithm that only works with lab-generated data. It must be adaptable to the environment and resistant to adversarial input. This should be the goal of all machine learning systems—to be robust against both natural and adversarial noise.

The operators of machine learning systems must be aware of their limitations and should have a general understanding of the ML model. 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 of their ML system and avoid the consequences of undetected noise.

1.2 Research Goals

Before implementing machine learning systems onto defense platforms such as submarines, we must fully understand their strengths, weaknesses, and vulnerabilities. This project aims to thoroughly analyze these qualities in a simulated microgrid called the 9-bus model [4]. We focus on robustness, a broad term that encompasses a system's ability to perform well in the presence of noise and uncertainties.

First, we design and train a simple feedforward neural network using MATLAB's neural network toolbox. This network serves as a baseline for the following experiments, since it performs well under perfect conditions but has no additional robustness considerations. Next, we add three types of noise (uniform random, Gaussian, and adversarial) to the testing data and measure the network performance. Then we test other network structures and measure their performance and stability against added noise. Finally, we train 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 determine the optimal structure for a machine learning tool using power grid data. Analysis of the average errors over large testing data sets shows the limits for the amount of random and adversarial noise that can be added while maintaining accuracy. To conclude, we provide recommendations for the training and usage of a machine learning tool along with parameters for which the tool can be trusted.

Overall, this analysis furthers our understanding of machine learning robustness and how to measure it. The methods and conclusions can be applied towards more complex data sets and real-life applications, such as submarine power grids. With this understanding, we can ensure that ML systems are only used in scenarios where their results can be trusted.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

2.1 Literature Review

Machine learning is currently being applied to a wide range of topics, including cyber security. Additionally, some methods exist for using ML to analyze power systems, however, their robustness is still in question. The following literature review covers the current research into general machine learning topics, methods of analyzing robustness, and applications to power systems.

2.1.1 Applications of Machine Learning for Cyber Security

The goal of any cyber defense system is to maintain data confidentiality, integrity, and availability [5]. Threats such as phishing, spam, and malware can all affect the users' experience. Traditional defenses such as firewalls and antivirus software work by following a predesigned set of rules, so they only protect against previously known and expected threats. Recently, machine learning has been applied to cyber security systems because it can learn from examples and adapt to new attacks [5]. Additionally, its flexibility and adaptive nature can detect slight variations in attacks that a more traditional system might overlook.

Machine learning does have some limitations, however. Huge data sets are required for adequate training, and ML models must be designed with specific threats in mind [5]. Trustworthiness is also an issue, since ML techniques often lack user transparency. More importantly, machine learning methods are vulnerable to noise or adversarial input, which can affect the training process or produce incorrect results when in use.

2.1.2 Robustness

There are many definitions of robustness for machine learning, all of which can be summarized as "the degree to which a system or component can function correctly in the presence

of invalid inputs or stressful environmental conditions" [6]. In other words, it is the ability of an ML model to perform well against noisy or adversarial input data.

O'Mahony et al. focus on an adversarial attack and define the stability of prediction as a robustness measurement [6]. Their definitions are given below.

"*Definition* 4.3. For each user-item pair $(a, j) \in A$, the *prediction error*, E_p, of prediction pre- and post-attack T is given by

$$E_p(a, j, T) = p'_{a,j} - p_{a,j} \tag{2.1}$$

where $p_{a,j}$ and $p'_{a,j}$ are pre- and post-attack predictions respectively" [6].

"*Definition* 4.4. The stability of prediction (SOP) of the set A to an attack T is given by

$$SOP(A, T, \alpha) = 1 - \frac{1}{|A|} \sum_{a \in A} \kappa_{a,j}(\alpha) \tag{2.2}$$

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" [6].

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.

Hembram et al. studied robustness in a gully erosion prediction system and defined their version of robustness as a combination of discrimination ability and reliability [7]. "Dis-

crimination ability refers to the separation capability of a model between gully presence and absence areas, and reliability refers to the agreement between observed gully locations and predicted gully locations" [7]. The paper's analysis provided several methods of measuring both aspects of robustness. For discrimination ability, efficiency, the Jaccard index, Matthew's correlation coefficient, the Kappa coefficient, and receiver operating characteristics were used. The reliability measurements were the root mean square error (RMSE) and the mean absolute error (MAE). These are common statistics equations that can be used to compare a variety of ML models. As shown in the gully erosion study, together they can give a detailed picture of the ML predictions, however, each measurement focuses on only one aspect of the problem. Most commonly, the RMSE is used as a benchmark error measurement.

Derks et al. performed a robustness analysis on two types of neural networks: radial base function (RBF) and multi-layered feedforward (MLF) networks [8]. The study generated noisy data sets by sampling noise from a normal distribution such that the error ratio was around 1‰, then 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, the researchers concluded that the RBF model was more robust. This example shows how noise can be added to input data and ways to measure its effect. Their results were specific to the data sets used, however, their methods could be applied to other ML models in order to test robustness.

2.1.3 Robustness Training Methods

Much of the literature on robustness focuses on adversarial attacks and the ability to protect against small perturbations which could cause classification errors [9]. Given a classification task with data drawn from a joint distribution $(x, y) \sim D$, with $x \in \mathbb{R}^d$ and $y \in \{0, 1, ..., k\}$ 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" [9]. This is formulated as a min-max problem, where the goal is to minimize the risk over D of a maximized perturbation $\delta \in \Delta$:

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

The problem with adversarial training is that this method often fails to protect against natural variation in the data. Rather than focus on adversarial attacks, Robey et al. propose a method of model-based robust deep learning to account for this natural variation [9]. 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' [9]. Finding this model of natural variation can be difficult since its geometry is often more complex than that of adversarial perturbations [9]. In fact, the perturbation-based adversarial training method is a special case of the model-based method, where $G(x, \delta) = x + \delta$ for $\delta \in \Delta := \{\delta \in \mathbb{R}^d : ||\delta|| \leq \epsilon\}$, for some small $\epsilon > 0$. For image classification problems, a known model of natural variation for rotation is $G(x, \delta) = R(\delta)x$ where $R(\delta)$ is the rotation matrix of angle $\delta \in \Delta := [0, 2\pi]$. 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 containing the original data and B containing the data with natural variation. Then the goal is to find a model G which transforms the distribution of data in A into the distribution of data in B.

Similar to the adversarial perturbation-based method, the goal of the model-based robustness training algorithm is to minimize the risk against the datum $x' := G(x, \delta*)$ generated by the worst-case nuisance, $\delta*$ which maximizes the loss values under the current weights, w:

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

This optimization problem is often difficult to solve exactly, but the problem can be modified to a finite-sample setting. Assuming there are a finite number of independent and identically

distributed samples from $D_n := \{(x^j, y^j)\}_{j=1}^n$, the min-max problem can be stated as

$$\min_w \frac{1}{n} \sum_{j=1}^{n} \left[\max_{\delta \in \Delta} l(G(x^j, \delta), y^j; w) \right].$$

The authors propose three variants of this algorithm, each of which add data generated by G to the original training set D_n [9]. First, the model-based robust training (MRT) method "randomly queries G to generate several new data points and then selects those generated data that induce the highest loss in the inner-maximization problem." The model-based adversarial training (MAT) algorithm "employs a gradient-based search in the nuisance space Δ to find loss-maximizing generated data." Finally, the model-based data augmentation (MDA) method "augments the training set with generated data by sampling randomly in Δ" [9].

These model-based robustness methods were compared to a standard training with no robustness considerations and an adversarial training with the perturbation-based method [9]. Their results show that the model-based methods provide significant robustness improvements to a variety of datasets and nuisances. A model of variation trained on a dataset D can be reused for training of classifiers on a different dataset D' and models of natural variation can be combined to improve robustness against a multiple 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, however, the main challenge is defining that model. Once an accurate model of variation is found, any of the three model-based training methods would provide significant robustness improvement.

2.1.4 Power Grid Disturbances

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 [3]. 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) [3]. 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 showed the best results, proving that using machine learning is a viable approach to power system disturbance classification.

Another study used data from phasor measurement units (PMU) within the power system to classify disturbances [10]. In that study, three classes were used: no event, natural event, and intrusion event. The method started by "manually constructing new features from the original data in the dataset so as to enhance data dimension." This was done to improve flexibility and gain higher accuracy and robustness in the model. Next, the data was split and training occurred by sorting the original features according to importance and then selecting diverse proportions of those features. This reduced the error from bad PMU measurements. Finally, the 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 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.

2.2 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 variations and 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 the loss function, or difference between the predictions and actual values.

This paper will focus 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 image in Figure 2.1. It 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 [11].

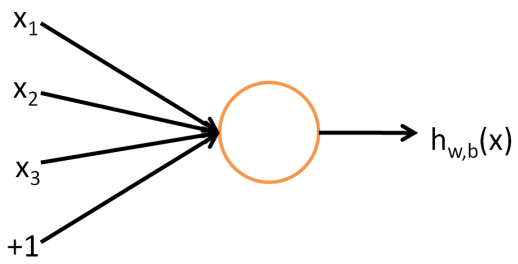

Figure 2.1. A single neuron.

Common activation functions are graphed in Figure 2.2. The sigmoid function is

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

The hyperbolic tangent, or "tanh" function, is

$$f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.3}$$

The rectified linear (ReLU) function is

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

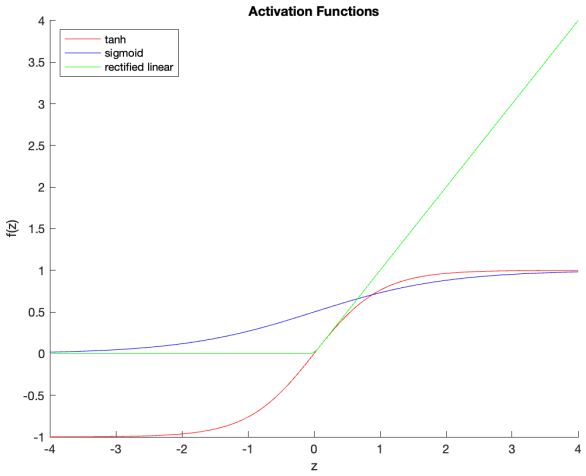

Figure 2.2. Common activation functions.

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 2.3 takes in data $x \in \mathbb{R}^3$ and outputs $y \in \mathbb{R}^2$. There are two hidden layers, the first with three neurons and the second with two.

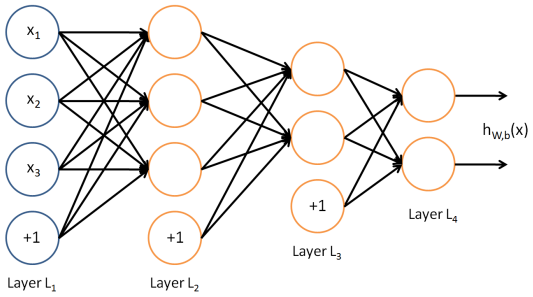

Figure 2.3. A neural network example.

The neural net uses parameters (W, b) where $W_{ij}^{(l)}$ denotes the weight of the connection between unit j in layer l and unit i in layer $l+1$. The parameter $b_i^{(l)}$ 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 $a_i^{(l)}$. 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 paper, we will consider 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:

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \cdots \\ P(y = K|x; \theta) \end{bmatrix} \tag{2.4}$$

where θ are the parameters of the model [11]. This is used for classification networks where the prediction should be one of K categories.

To train the neural network we use the backpropagation algorithm. 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 paper will focus primarily on feedforward multi-layered neural networks, also called deep neural networks (DNN). However, there are many other forms of machine learning algorithms and network structures which may be topics for further study.

2.3 Power System Modeling

To train and test a machine learning method, we first need large quantities of data. To start, we create a model power grid and gather simulated data. Our system is based on the classical 9-bus dynamical microgrid model, as described by Anderson and Fouad [4]. It contains three

generators (denoted by i= 1, 2, 3) and is modeled with "6 state variables, rotor angle (δ_i) and angular velocity (w_i), and 12 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 ordinary differential Equations (2.5-2.7).

$$\frac{2H_i}{\omega_R}\frac{d\omega_i}{dt} + D_i\omega_i = P_{mi} - P_{ei}\ ; \quad i = 1, 2, 3 \tag{2.5}$$

$$\frac{d\delta_i}{dt} = \omega_i - \omega_R \tag{2.6}$$

$$P_{ei} = E_i^2 G_{ii} + \sum_{j=1, j\neq i}^{n} E_i E_j Y_{ij} cos(\theta_{ij} - \delta_i + \delta_j) \tag{2.7}$$

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} [4].

A drawing of the classical 9-bus model is shown in Figure 2.4.

2.3.1 Simulations

We implemented the 9-bus system described above in MATLAB. Then we simulated trajectories for normal operations and faults at each generator.

We compiled the data into two sets, training and validation. Each contains 18000 examples with data $x \in \mathbb{R}^{30}$ 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 are 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.

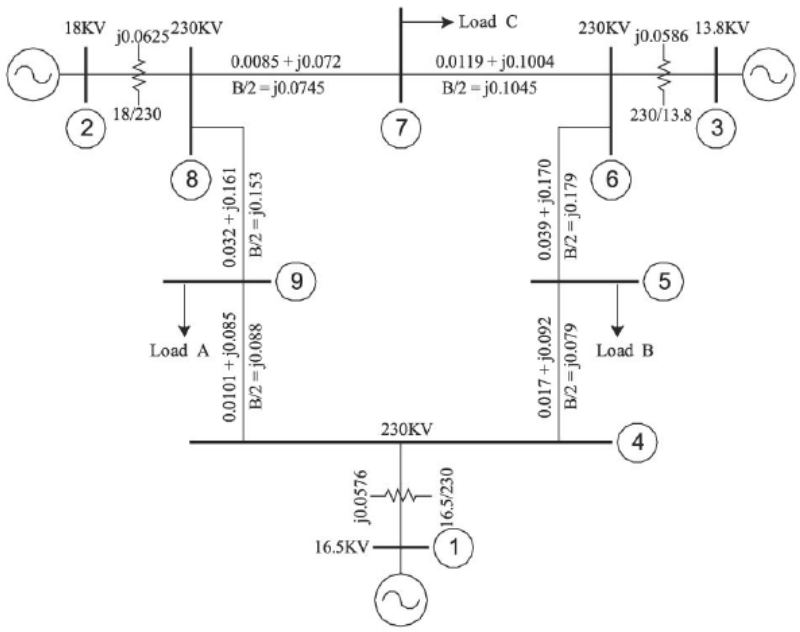

Figure 2.4. The 9-bus model. Source: [12].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

3.1 MATLAB Neural Network Tools

To build and train our neural networks we utilized MATLAB's nntraintool (see Figure 3.1). The simple command "network" creates a custom shallow neural network, where the user can define the number of inputs, layers, and connections. Then we can 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 [13]. Each hidden layer uses the hyperbolic tangent activation function. For the parameters, our baselines were 500 training iterations, a gradient of 10^{-7} and 100 validation checks. Once the network is trained, it can be saved and used later to make predictions from different input data. An example code is provided in Appendix A.1.

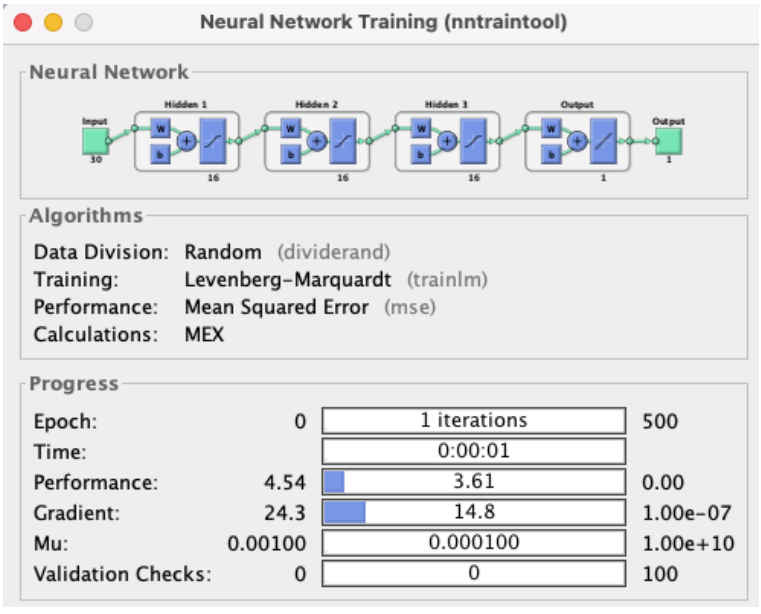

Figure 3.1. MATLAB's nntraintool for training neural networks.

3.2 Network Structure

The neural network tool in MATLAB contains several basic network structures. The first 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. A FFN with 3 hidden layers of 16 nodes each is shown in Figure 3.2.

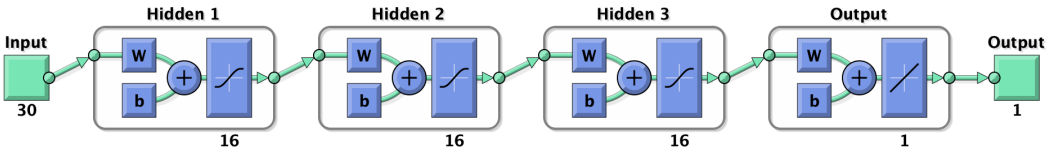

Figure 3.2. Example feedforward neural network.

We tested both the *feedforwardnet* for regression and the *patternnet* 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 (2.3) 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 (2.4) for its output. Although very similar, these small differences cause noticeable variation in robustness, as described more in chapter 4.

The second network we tested is the cascade forward net (CFN). This structure is similar to a feedforward network but includes more connections, with each layer connected to the input and every previous layer. Each hidden layer uses the hyperbolic tangent (tansig) activation function and the output layer uses the pure linear (purelin) function. As before, we can customize 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 density of connections. An example cascade forward net with 3 hidden layers of 16 nodes each is shown in Figure 3.3.

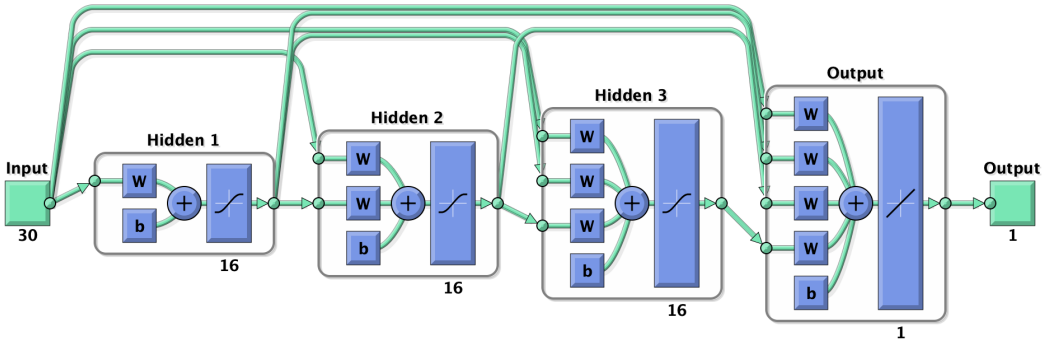

Figure 3.3. Example cascade forward network.

3.3 Added Noise

Starting with a network trained on the "clean" data (generated by the 9-bus simulation), we added various forms of noise to our validation data set and recorded 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 normalized data. For example, the process of adding uniform random noise (Z) to a normalized data vector, $\mathbf{x}_{norm}$, is calculated as:

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

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

An example code for this method is provided in Appendix A.2. Since the networks were trained on a "clean" data set; i.e., no added noise, the "noisy" data is different from the training set and therefore poses a challenge to the robustness of the network. Section 4.2 describes the effects of this random noise.

3.4 Gradient-Based Perturbations

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 targeted 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 very small, typically 10^{-5}. This is repeated for each index, i, of x. Since our input data $\mathbf{x} \in \mathbb{R}^{30}$, this gradient $\nabla(h_{W,b}(\mathbf{x})) \in \mathbb{R}^{30}$.

The noise vector (Z) is computed for each data point, x, by calculating $\nabla(h_{W,b}(\mathbf{x}))$ and normalizing it. This noise is then multiplied by the noise scalar and added to the normalized data, as shown below. Example code is provided in Appendix A.3.

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

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

The finite difference method is a slow process, therefore we tested the gradient-perturbations on smaller data sets with $N = 1000$ test values. To ensure representative results, before each test we shuffled the validation set and then selected the first N data points.

3.5 Robustness Measurements

As shown in the literature, there are numerous ways of measuring the robustness of a ML method. For this research we focus on the average error, stability of prediction (SOP), and percent misclassified. After each network test we first calculate 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 [6]. This is calculated using the prediction error, E_p. Equations 2.1 and 2.2 show the calculations for these measurements. 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, represents the added random or gradient-based noise.

The percent misclassified is used to compare the classification networks (patternnet structure) to the regression networks (feedforward and cascade forward). To do this, we round the regression output to the nearest integer. Then the error measurement is the percent of data points with the wrong predicted value ($y \in \{0, 1, 2, 3\}$).

3.6 Gathering Real Data

The majority of this project focuses on simulated data from a 3-generator 9-bus microgrid model (Section 2.3.1), however, the clear follow-on is to apply the same robustness measurement techniques to real data. To do this, we set up a Frequency Disturbance Machine (FDR) as part of the University of Tennessee's FNET/GridEye project [14]. Our system is composed of the FDR and a small air compressor connected to the same conventional 120V power outlet. These components are shown in Figures 3.4 and 3.5. By running a program through MATLAB we can collect the frequency output of the outlet over a given time interval.

Figure 3.4. FDR machine.

Figure 3.5. Air compressor.

We collected two types of data: normal operations and powering on the machine. For the first, the system records the frequency output over three minute intervals while nothing is turned on. Second, we gathered data for three minute intervals where the air compressor is repeatedly turned on for 5 seconds, then off for 10. To prepare the data for use in a neural network, we randomly select 12-second intervals and form our input vectors, $x \in \mathbb{R}^{120}$ of the frequency output from the FDR. Each x is assigned either a value of $y = 0$ for normal operations, or $y = 1$ if the air compressor is turned on.

The training and testing sets each contain 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 3.1.

CHAPTER 4:
Results

4.1 Structure of a Feedforward Neural Network

The first goal of this analysis was to determine the most robust configuration of a basic feedforward neural network (FFN). Working off the connection scheme described in section 3.2, we varied the number of layers and the number of hidden nodes per layer to determine the best combination to balance accuracy and simplicity.

First, we started 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. These results are shown in Figure 4.1.

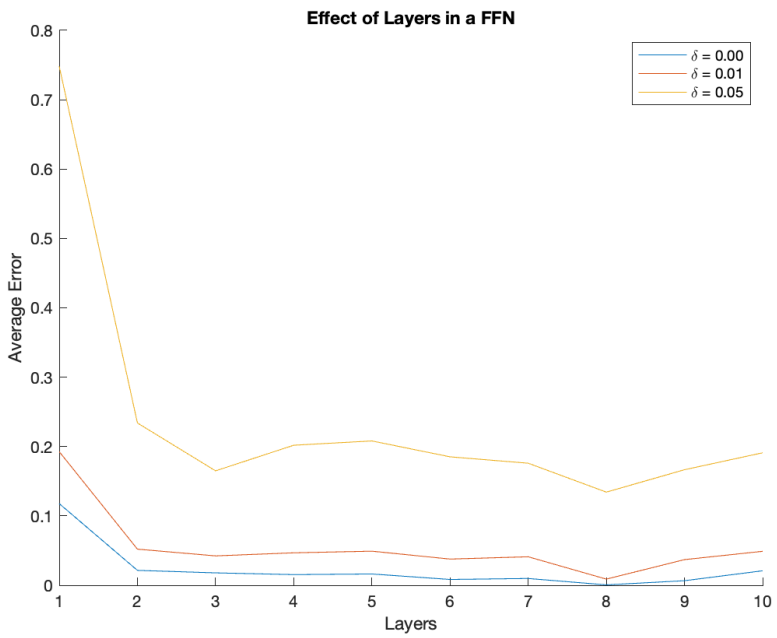

Figure 4.1. Effect of number of layers in FFN with 16 hidden nodes each, tested against uniform noise.

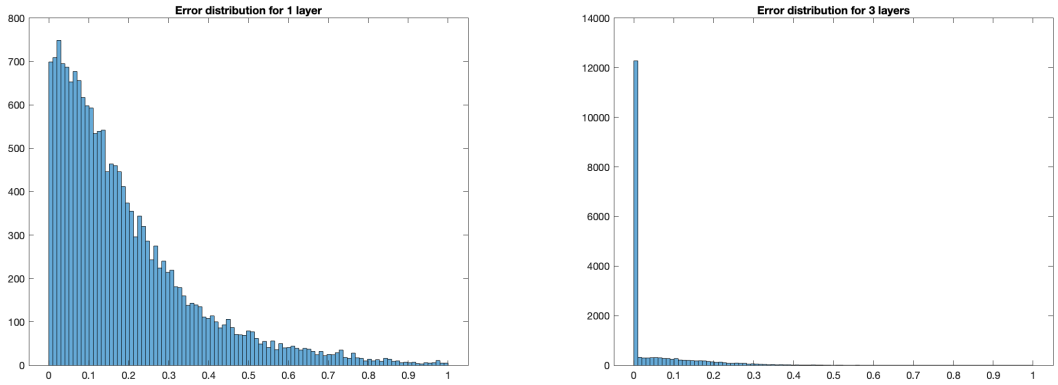

Figure 4.2. Error distribution for 1 and 3 layer FFNs against testing data with added uniform noise of $\delta = 0.01$.

Figure 4.2 shows how the errors are distributed over the entire testing set, resulting in the average errors plotted in Figure 4.1. As these graphs show, the average error evens 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 increases drastically. For example, using a MacBook Pro with the M1 chip (8 core CPU), a 3-layer FFN takes roughly 13 minutes to train, 6 layers takes 36.5 minutes, and 9 layers takes 1 hour and 28 minutes. Therefore, we can conclude that 3 layers is sufficient for good results with a reasonable training time requirement.

Next, we vary the number of nodes per layer of a 3-layer FFN. Once again, we tested the networks against data with added uniform noise. Figure 4.3 shows the average error of networks with 4, 8, 12, 16, or 20 nodes per layer as the noise scalar increases.

This data shows that the optimal number of nodes per layer is 16. As expected, less than 8 nodes does not produce a very accurate or robust network, but it may be surprising to see that 20 nodes also perform worse than 16. This may be a result of overparameterization. Since more nodes per layer also 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 time (about 13 minutes).

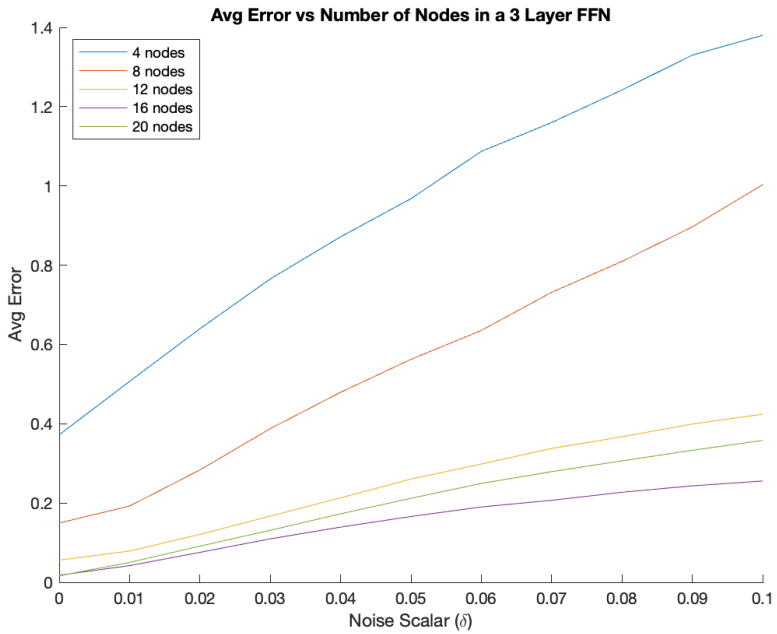

Figure 4.3. Effect of number of nodes per layer in FFN with 3 layers, tested against uniform noise.

4.2 Types of 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 effects of different types of noise. We added uniformly random ($U[-\sqrt{3}, \sqrt{3}]$) noise, normal ($N(0,1)$) noise, and gradient perturbations to the input data, then ran this altered data through the same FFN. Note that this network was trained using "clean" data (i.e., no added noise), so each of the three altered data sets presents a new challenge to the network. Figure 4.4 shows the results of this study, with the degree of noise (noise scalar) vs the average error of predictions. Figure 4.5 shows the error distributions for each type of noise with $\delta = 0.05$ and Table 4.1 provides the numerical values for the cumulative distribution.

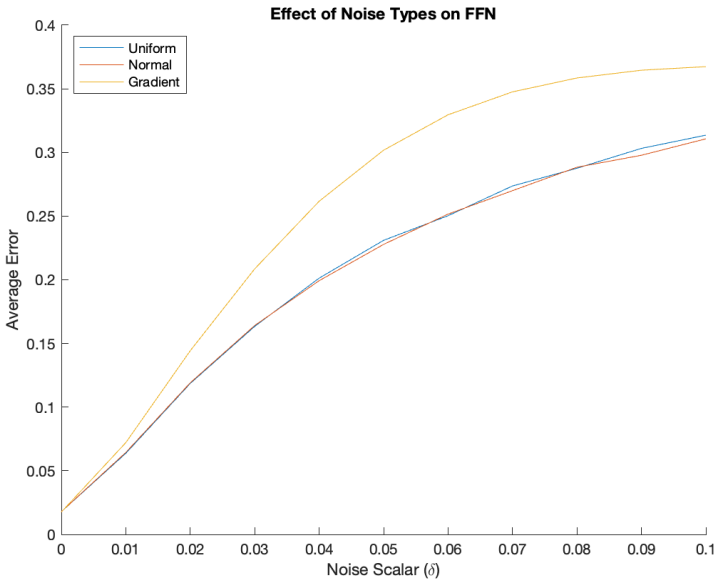

Figure 4.4. Effect of different noise types on a 3-layer, 16-node FFN.

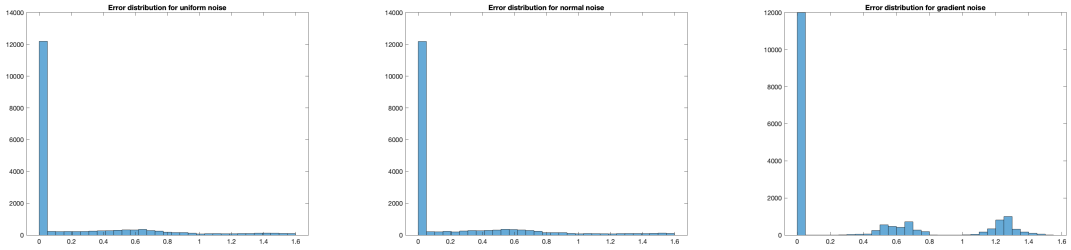

Figure 4.5. Error distribution for a 3 layer, 16 node FFN against uniform, normal, and gradient perturbations ($\delta = 0.05$).

	< 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 4.1. 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 as long as the mean ($\mu = 0$) and standard deviation ($\sigma = 1$) are the same.

When looking at the error distributions, however, it is clear that the vast majority of data points are classified correctly, regardless of the noise type. The difference is that the gradient noise causes significantly more noise 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 since 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.

4.3 Network Type

Next we move on to different types of networks, testing 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 3.2. 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 4.6 and 4.7.

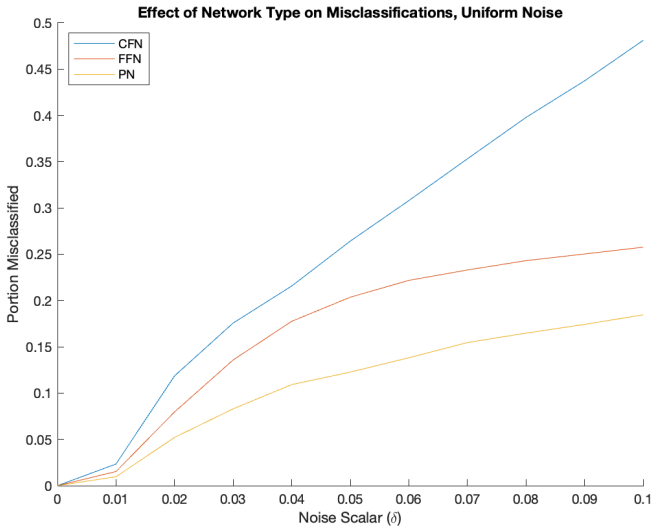

Figure 4.6. Comparison of network types against uniform noise.

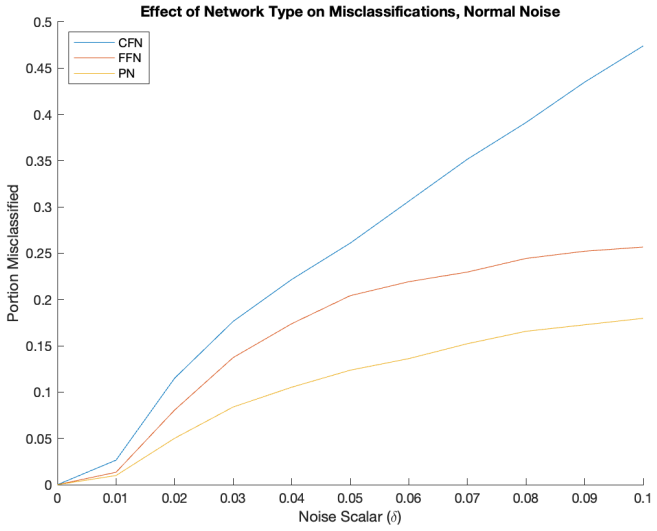

Figure 4.7. Comparison of network types against normal noise.

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 and the cascade forward net produces the

most. This is likely because 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.

4.4 Network-Noise Profiles

For further comparison between networks, we graphed their stability of prediction vs added noise of each type (uniform, normal, and gradient). Figure 4.8 shows the data for random noise and Figure 4.9 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. Between the two types of random noise we see virtually no difference, once again confirming that the different distributions have the same effect so long as their mean and standard deviation are identical. The gradient perturbations, however, have a clearly different pattern emerging 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.

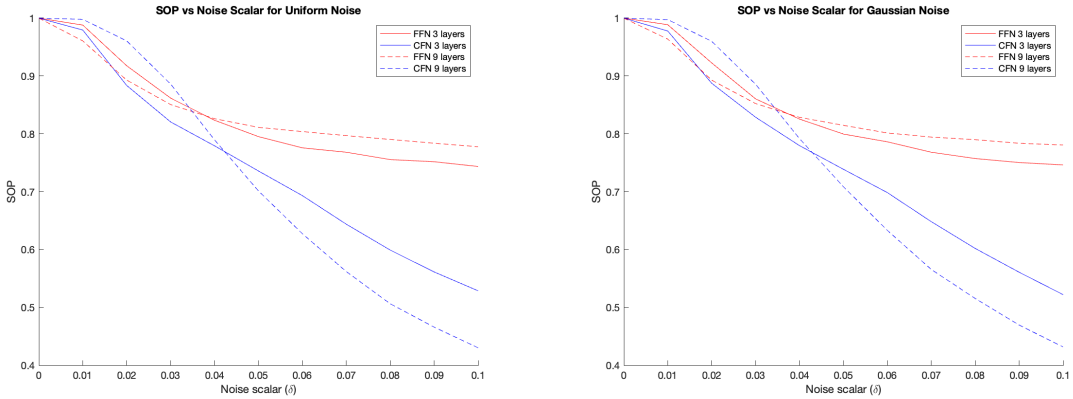

Figure 4.8. SOP comparison for 4 network structures with random added noise.

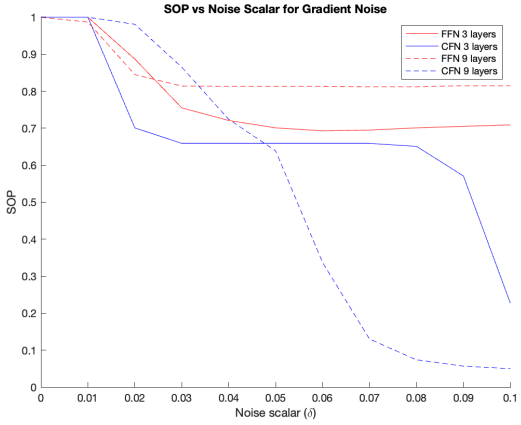

Figure 4.9. SOP comparison for 4 network structures with gradient perturbations.

4.5 Training 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 (Section 3.3) to the original training data set (Section 2.3.1). 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 4.10, 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).

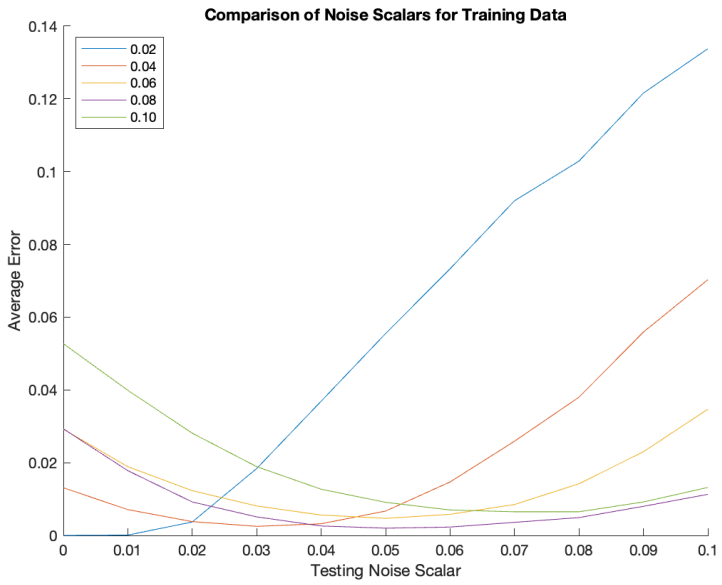

Figure 4.10. Effect of uniform noise added to the testing data.

4.6 Real Data

Following the same procedure as Section 4.1, we train several FFNs with structures that vary in number of layers and nodes per layer. Figure 4.11 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). Unlike Figure 4.1, here 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 as the layers increase.

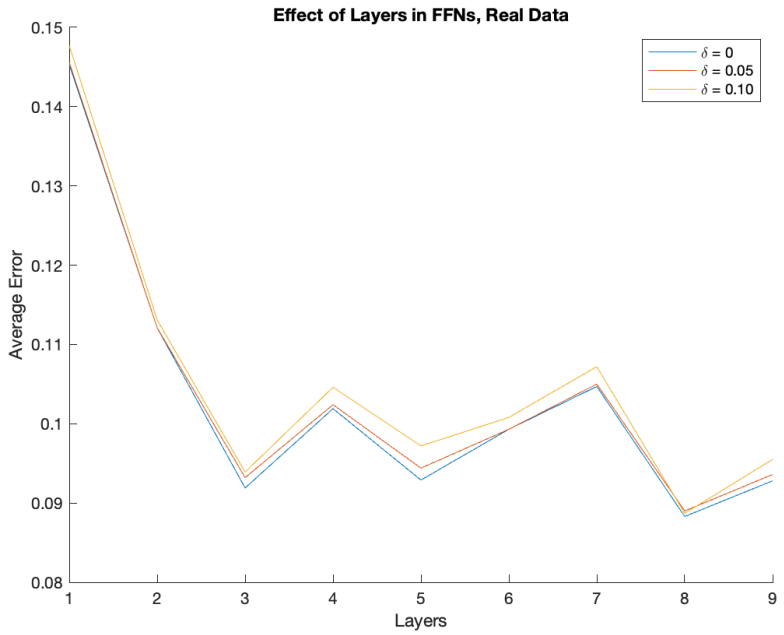

Figure 4.11. Effect of number of layers in a FFN, each with 16 nodes per layer.

Figure 4.12 shows the average error compared to the noise scalar for several 3-layer networks with different number of nodes per layer. Again, we see that the noise scalar seems to have very little effect. The number of nodes, however, mirrors Figure 4.3 with 16 showing the lowest error.

While further study is certainly required for this real data set, we do see interesting comparisons with the simulated data results from Section 4.1. For both, the optimal configuration appears to be 3 layers with 16 nodes per layer. With the real data, however, added noise does not have much of an effect like it does for the simulated data. This could be due to the already "noisy" nature of real data, whereas the simulated data is more "ideal" from the start.

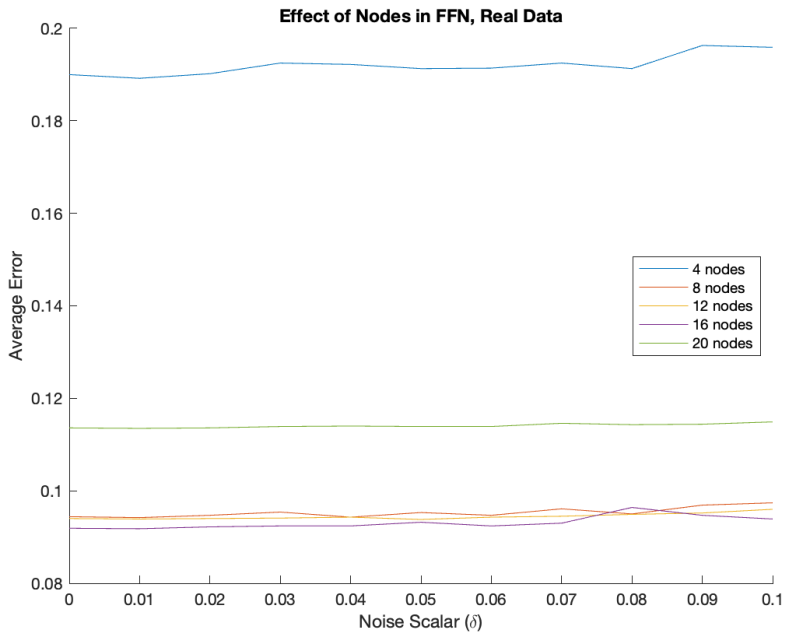

Figure 4.12. Effect of number of nodes per layer in a FFN, each with 3 layers.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion

5.1 Analysis

This project aimed to analyze the strengths, weaknesses, and vulnerabilities of machine learning systems to gain insights into their robustness. To this end, we used MATLAB's Deep Neural Network Toolbox and trained networks to predict faults in a simulated microgrid. These networks were challenged by adding both random and gradient 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.

Based on analysis of the network structure, we conclude that 3 layers and 16 nodes per layer is sufficient for this data set. Figure 4.1 shows diminishing returns past 3 layers and Figure 4.3 shows that 16 nodes per layer achieves the minimum average error across all noise levels. 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. Counter to intuition, 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 drastically 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 ($U(-\sqrt{3}, \sqrt{3})$) and normal ($N(0,1)$) 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 was not the case. Instead, we conclude that noise drawn from any distribution with the same mean and standard deviation will produce the same result in the network.

Adding adversarial noise in the direction of the network gradient did have a significantly different effect from the random noise distributions. We see this in Figure 4.4 and Figure 4.5.

This gradient perturbation using the finite difference method is a simple form of adversarial attack, so we can conclude that more complex attack methods will cause greater error in the ML system.

Figure 4.6 shows how three different network types perform against varying levels of uniform noise. This data agrees with our first conclusion, that more complex networks (i.e., the CFN with more connections between layers) does not necessarily result in a more robust network. 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 (SOP) 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. Interestingly, the CFN with 3 layers is more stable than with 9 layers, and the FFN achieves levels out around $\delta = 0.04$ for all noise types. This implies that a higher noise scalar will not affect the network any more negatively. While these SOP graphs do not present clear conclusions about robustness, they do show interesting patterns that may have future applications for identifying when noise is present in a system.

Finally, we added uniform random noise to the training data as a way to improve the robustness of our networks against noisy testing data. Figure 4.10 shows that we achieve the lowest error when a network is 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.

5.2 Future Work

The next step for this work is to continue applying the robustness measurements and improvement techniques to real data. Section 4.6 begins to analyze the predictions for turning on/off an air compressor, however, this is simply proof of concept. Collecting and testing FDR data from larger, more complex power grids will further develop the methods and strengthen our conclusions about neural networks applied to this type of dynamic data.

Alternatively, future work could continue to use the simulated data and expand the results for other types of noise. More complex distributions of random noise or more pointed adversarial attacks could provide useful insight into how the networks behave under harsh conditions.

Overall, the end goal is to further our understanding of machine learning robustness for power grid data so that these concepts can be used in real-world applications, such as naval vessels and military bases. Future work should be focused on adding complexity to the experimental design such that the data and conclusions more closely resemble these applications.

5.3 Applications

Overall, this work demonstrates the applicability of neural networks for dynamic power grid systems and provides a series of tests for understanding their robustness. These methods of measuring robustness can be applied to any other form of machine learning and other data sets.

Specific to power grids, the use of machine learning could aid in oversight and maintenance. Our neural networks were able to accurately predict if a fault occurred and where it happened. Even with significant added noise, the average error remained beneath 0.5, the criteria for misclassification. When applied to real power grids, the neural networks could assist human operators in quickly identifying faults or unusual behavior before the problem shut down the grid. On submarines, naval bases, and other mission-essential platforms, this early warning from ML systems could be essential in maintaining operability and scheduling maintenance.

Further analysis of the effects of added noise and adversarial effect can also aid in the defense of machine learning systems. If several neural networks with various structures are simultaneously processing incoming noise, any differences in their predictions could indicate the presence of noise. This concept is demonstrated in the SOP noise-network profiles of Section 4.4. Assuming the group of networks perform identically under ideal conditions, the magnitude of the discrepancy between network predictions can determine the amount and type of noise present.

From an offensive perspective, neural networks could be used to gain insight into adversarial operations. If one could gather frequency data from an adversary's power grid using remote or mobile sensors such as the Frequency Disturbance Recorder, an ML system could analyze the data and draw conclusions about the usage of the grid. Similarly to how the neural network could determine when the air compressor is turned on, a similar ML system could determine if and when an adversary powers up a large machine such as a missile system.

For both maintenance and surveillance applications, the methods and insights presented in this paper should guide robustness evaluations of the machine learning systems. Testing various network structures to find the optimal configuration and training with more noise than is expected operationally will improve the resilience and accuracy of neural networks, while groupings of networks can help identify the presence of noise.

APPENDIX: MATLAB Example Code

A.1 Train a Neural Network

```
% Train network using MATLAB Deep Learning Toolbox

%% Load and Prepare Training Data
load data_train.mat
Xt = X;
Xt_norm = normalize(Xt, 2);
Yt = Y;

%% Build Network
net = feedforwardnet([16, 16, 16], 'trainlm'); %levenberg-marquardt training method
net.trainParam.epochs = 500;   %max epochs
net.trainParam.max_fail = 100; %max validation checks

%% Train Network
[TrainedNet, tr] = train(net, Xt_norm, Yt);
save feedforwardnet_3layer_16nodes

%% Test Network
load data_val.mat
testX = normalize(X,2);
testY = Y;

Ypred = TrainedNet(testX);

%% Calculate Accuracy
Z = abs(Ypred - testY);
avgerr = mean(Z);

Zround = abs(round(Ypred) - testY);
N = nonzeros(Zround);
mispredictions = sum(N);
accuracy_rounded = 1 - sum(N)/length(Zround);
```

A.2 Add Noise to Training Data

```
% Add noise

Noise = randn(size(Xt)); %Gaussian noise
d = 0.1; %noise scalar

Xt_perturbed = Xt_norm + d*Noise;
```

A.3 Finite Difference Method for Gradient Perturbations

```
% Gradient Perturbation
N = 1000;
h = 10^(-5);

%Shuffle and Shorten
order = randperm(size(testX, 2));
testX_shuff =  testX(:, order);
testY_shuff = testY(order);
testX_short = testX_shuff(:, 1:N);
testY_short = testY_shuff(1:N);

% Calculate Gradient and perturb
Ddx = zeros(size(Xk_short));
testX_pert = zeros(size(testX_short));
noise = zeros(size(testX_pert));

for i = 1:N
    x0 = testX_short(:, i);
    Ddx(:,i) = finitediff(net, x0, h);
    noise(:,i) = Ddx(:,i)./norm(Ddx(:,i));
end
```

```
function [ddx] = finitediff(Net, x0, h)
%Approximate the gradient at x0
ddx = size(x0);

for i = 1:length(x0)
    xb = x0;
    xb(i) = xb(i) - h;
    xf = x0;
    xf(i) = xf(i) + h;
    ddx(i) = (Net(xf) - Net(xb))/(2*h);
end

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] Connecticut Department of Energy and Environmental Protection, "Microgrid grand and loan programs," Jan. 2020 [Online]. Available: https://portal.ct.gov/DEEP/Energy/Microgrid-Grant-and-Loan/Microgrid-Grant-and-Loan-Program

[2] E. Wood, "Microgrid project at submarine base receives $5m connecticut grant," Microgrid Knowledge, Sept. 10, 2018 [Online]. Available: https://microgridknowledge.com/microgrid-project-submarine-base/

[3] R. C. B. Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, "Machine learning for power system disturbance and cyber-attack discrimination," in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. IEEE, 2014, pp. 1–8.

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

[5] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, "A survey on machine learning techniques for cyber security in the last decade," *IEEE Access*, vol. 8, pp. 222 310–222 354, 2020.

[6] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 4, pp. 344–377, 2004.

[7] T. K. Hembram, S. Saha, B. Pradhan, K. N. Abdul Maulud, and A. M. Alamri, "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, 2021.

[8] E. Derks, M. S. Pastor, and L. Buydens, "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, 1995.

[9] A. Robey, H. Hassani, and G. J. Pappas, "Model-based robust deep learning," *arXiv preprint arXiv:2005.10247*, p. 2, 2020.

[10] D. Wang, X. Wang, Y. Zhang, and L. Jin, "Detection of power grid disturbances and cyber-attacks based on machine learning," *Journal of information security and applications*, vol. 46, pp. 42–52, 2019.

[11] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang, and S. Tandon, "UFLDL tutorial," accessed Apr. 26, 2022 [Online]. Available: http://ufldl.stanford.edu/tutorial/

[12] K. Sun, J. Qi, and W. Kang, "Power system observability and dynamic state estimation for stability monitoring using synchrophasor measurements," *Control Engineering Practice*, vol. 53, pp. 160–172, 2016.

[13] MathWorks Help Center, "trainlm Levenberg-Marquardt backpropagation," accessed May 10, 2022 [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/trainlm.html

[14] University of Tennessee Knoxville, "Fnet/grideye web display," May 20, 2022 [Online]. Available: http://fnetpublic.utk.edu

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California