NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANOMALY DETECTION FOR THE NAVAL SMART GRID SYSTEM USING AUTOENCODER NEURAL NETWORKS

by

Preston C. Musgrave

June 2022

Thesis Advisor:	Preetha Thulasiraman
Second Reader:	Murali Tummala

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 ANOMALY DETECTION FOR THE NAVAL SMART GRID SYSTEM USING AUTOENCODER NEURAL NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Preston C. Musgrave	
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)

In 2019, the Naval Facilities Engineering Command (NAVFAC) deployed its first smart grid infrastructure in Norfolk, VA, enabling shore commands to meet energy goals set by the secretary of the Navy. However, with increased functionality and control comes increased vulnerability to malicious cyber activity. This research aims to address anomaly detection using an autoencoder neural network as an intrusion detection mechanism on the NAVFAC smart grid. We built and experimented with multiple autoencoder structures to identify an optimal model that provides the best results in terms of precision, recall, and accuracy for the data sets used. We trained our autoencoder on NAVFAC-provided advanced metering infrastructure (AMI) data. We used the NAVFAC smart grid data set to simulate 14 different false data injection attacks (FDIA). Our experiments, performed with Python and TensorFlow, showed that an autoencoder is an effective instruction detection system (IDS) when the threshold is tuned correctly. Moreover, our results show that the activation function and optimizer used may affect performance. Thus, the "best" autoencoder depends on the customer's needs and the threat environment.

14. SUBJECT TERMS smart grid, cyber-security, machine learning, autoencoder, clustering, neural network	**15. NUMBER OF PAGES** 81
	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.

ANOMALY DETECTION FOR THE NAVAL SMART GRID SYSTEM USING AUTOENCODER NEURAL NETWORKS

Preston C. Musgrave
Lieutenant, United States Navy
BA, Fairmont State University, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE
(ELECTRICAL ENGINEERING)

from the

NAVAL POSTGRADUATE SCHOOL
June 2022

Approved by: Preetha Thulasiraman
Advisor

Murali Tummala
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In 2019, the Naval Facilities Engineering Command (NAVFAC) deployed its first smart grid infrastructure in Norfolk, VA, enabling shore commands to meet energy goals set by the secretary of the Navy. However, with increased functionality and control comes increased vulnerability to malicious cyber activity. This research aims to address anomaly detection using an autoencoder neural network as an intrusion detection mechanism on the NAVFAC smart grid. We built and experimented with multiple autoencoder structures to identify an optimal model that provides the best results in terms of precision, recall, and accuracy for the data sets used. We trained our autoencoder on NAVFAC-provided advanced metering infrastructure (AMI) data. We used the NAVFAC smart grid data set to simulate 14 different false data injection attacks (FDIA). Our experiments, performed with Python and TensorFlow, showed that an autoencoder is an effective instruction detection system (IDS) when the threshold is tuned correctly. Moreover, our results show that the activation function and optimizer used may affect performance. Thus, the "best" autoencoder depends on the customer's needs and the threat environment.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AMI	advanced metering infrastructure
Adam	adaptive moment estimation
ANN	artificial neural network
BAN	base area network
BCS	base computer application software
DOE	Department of Energy
DMZ	demilitarized zone
FDIA	false data injection attacks
HAN	home area network
IDS	intrusion detection system
KNN	k-nearest neighbor
MTU	master terminal unit
MSE	mean squared error
NAVFAC	U.S. Naval Facilities Engineering Command
NAN	neighborhood area networks
PCA	principal component analysis
PCS	process control systems
ReLU	rectified linear unit
RTU	remote terminal units

SGD stochastic gradient descent

SCADA supervisory control and data acquisition

PCS process control systems

USN U.S. Navy

WAN wide area network

Acknowledgments

I thank God for continuing to bless me with incredible opportunities to grow in my faith and my profession. Thank you for placing all the brilliant classmates and professors in my life, especially as we navigated through the COVID-19 pandemic. To Dr. Preetha Thulasiraman, thank you for your wisdom, guidance, and support. I could not have asked for a more committed and thoughtful advisor and professor. And last, to my wife, Kerry, thank you for always being by my side and my greatest champion. I cannot imagine going through this process without you.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

In response to energy modernization goals initially established by then-Secretary of the Navy Ray Mabus, the U.S. Naval Facilities Engineering Command (NAVFAC) developed smart grid infrastructure for select U.S. Naval bases to increase their energy security and reduce their energy consumption [1]. In 2019, NAVFAC began transitioning facilities from traditional power grid transmission infrastructure to this modernized smart grid [2].

The Department of Energy (DOE) describes the smart grid as a rebuild of the existing power grid by incorporating digital technology that enables two-way communication between the grid operator and its customers [3]. The objective of the smart grid is to increase power distribution efficiency, reduce operating costs, and provide quick restoration of services after power disturbances [3], [4].

Figure 1.1 illustrates the differences between the traditional grid and the smart grid. In a traditional grid, the power flows from the supplier to its consumers without a feedback mechanism. This architecture requires technicians to manually record data from on-location meters, a laborious and time-consuming process when troubleshooting issues [2]. In a smart grid, there is a two-way flow of information. Through the advanced metering infrastructure (AMI), information regarding the current state of the grid, predicted power demand, customer information, and other data are exchanged to allow the smart grid to tailor its use, increasing efficiency and responsiveness [2], [5]. The NAVFAC smart grid achieves the same objectives as the commercial smart grid, primarily through two means. First, the two-way communication feature enables the grid to tailor the power distribution to meet end-user demands, reducing costs while increasing reliability [4]. Second, the NAVFAC smart grid can remotely monitor its performance or troubleshoot outages through smart metering technology connected via a wide area network (WAN) [6]. This function helps NAVFAC maximize grid efficiency and reduce maintenance costs.

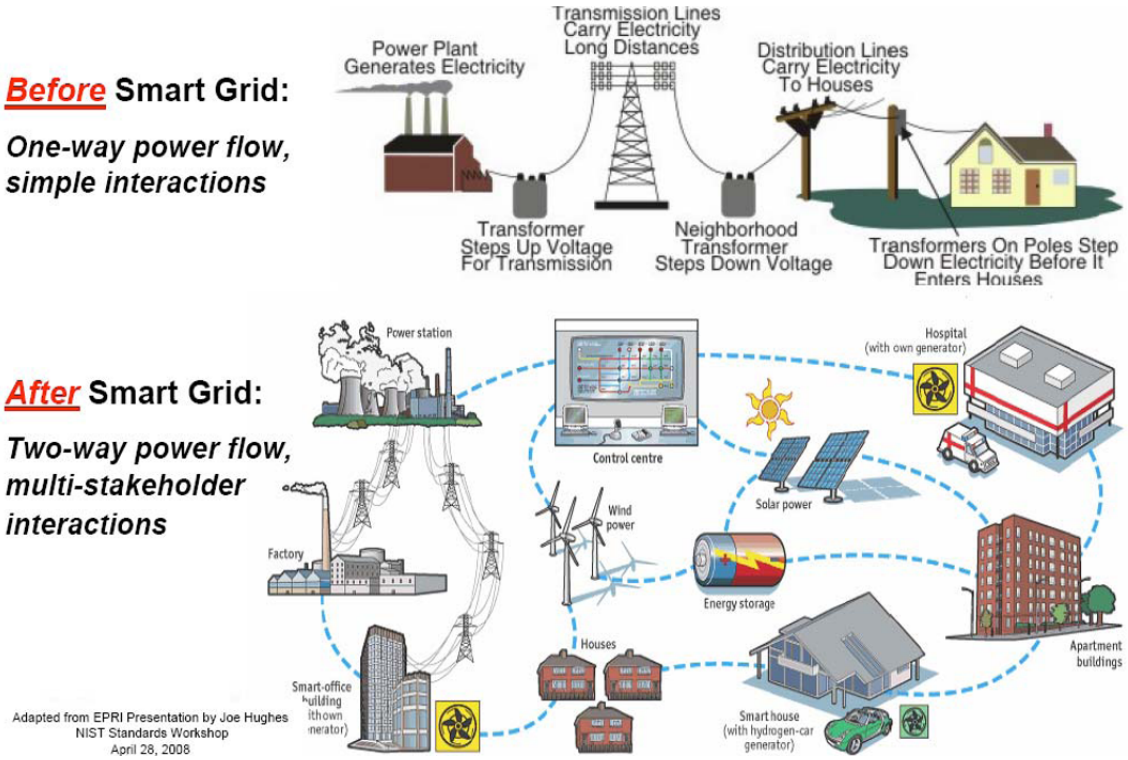

Figure 1.1. Power Infrastructure Design Comparison: Smart Grid versus Traditional Grid. Source: [10].

With the tremendous benefits of a smart grid comes a cost. Every connected device on the grid becomes a potential target for malicious activity, increasing the attack surface for nefarious cyber actors [6]. Once protected by isolation and proprietary technologies and protocols, power grids that transition to smart infrastructure become vulnerable to similar threats plaguing the Internet [7]. Additional considerations and security measures must be implemented that were not required in the past. An intrusion detection system (IDS) is one of many necessary additions [7].

1.1 Research Motivations

As a critical support node to the U.S. Naval fleet, the NAVFAC smart grid is a priority target for any malicious actor trying to disrupt ship movements or put a valuable asset at risk. If unmitigated, the NAVFAC smart grid vulnerability introduces an unacceptable risk to Navy

combat readiness. To defend against attacks, smart grid technology must be sophisticated enough to identify malicious activity and resilient enough to continue operating while implementing mitigation efforts.

Currently, the state of smart grid cybersecurity is component-specific. Due to the complexity of the smart grid network and numerous threats, researchers have not established a holistic approach to smart grid security [8]. In [9], the author presents five cybersecurity categories for smart grids – process control systems (PCS), smart meter security, power system state estimation security, smart grid communication protocol security, and smart grid simulation for security analysis. Our research overlaps with smart meter security and power system state estimation security.

Smart meter security issues include physical tampering, since the devices are at the end-user location, and inaccurate data entering or exiting the meter. Power system state estimation security focuses on the integrity of the data transmitted between the PCS and the smart meters. Mechanisms exist that can differentiate between corrupted data and normal data. However, most techniques fail to detect false data injection attacks (FDIA) [9]. An FDIA is malicious activity designed to manipulate data values. The techniques that do detect FDIAs require a labeled data set [10]. However, due to the difficulty of detecting FDIAs and their seldom occurrence in historical data, smart grid training data is highly unbalanced [11]. Our research provides a defense against FDIAs using unlabeled data, filling a gap in smart grid cybersecurity.

1.2 Contributions

In this thesis, we develop an IDS for the NAVFAC smart grid using unsupervised machine learning. Specifically, we use an autoencoder neural network to predict anomalous activity.

We chose an autoencoder because of its ability to "learn" the feature space of normal activity without using labeled training data. Due to the time-consuming nature of labeling data, it is unreasonable to expect a completely labeled data set from any operator of such an extensive network. The label-free aspect of unsupervised learning was therefore critical to our research.

Autoencoders provide an elegant solution to the data set labeling problem. During the

training process, the model ingests non-malicious data to determine a baseline threshold, which delineates between "normal" and "anomalous" data. In post-training, the autoencoder quickly identifies anomalous activity by comparing the reconstruction error of each input to the threshold established during training.

Our research used a NAVFAC data set to train an autoencoder. This data set was provided by the engineers and smart grid program manager at NAVFAC. The data set was generated by the NAVFAC's AMI, which is a configuration of smart meters, data management devices, and a two-way communication network between the central system and the meters [12]. To the best of our knowledge, this is the first time that anomaly detection research has been conducted using Navy smart grid data.

To test the autoencoder ability to detect anomalous activity, we introduced FDIAs by manipulating input values to evaluate the autoencoder performance. Thus, the contributions of this work are as follows:

- Design of a network intrusion mechanism based on an autoencoder neural network. We used and processed a novel, real-world NAVFAC data set for training.
- Provide an unsupervised machine learning framework that overcomes the unbalanced training data set problem in smart grid research.
- Evaluation of the developed autoencoder on 14 different data sets, representing varying degrees of FDIA to show the model performance.
- Compare the developed autoencoder with a deeper model given in [11] to show performance differences and tradeoffs.
- Demonstrate that autoencoders can recognize full-scale attacks (targeting every input parameter) and more nuanced attacks (targeting specific input nodes).

This thesis supports funded research through the Office of Naval Research to study enhancing the cyber resiliency of the Naval smart grid.

1.3 Thesis Organization

The remainder of this thesis is organized as follows: In Chapter 2, we discuss related research and the fundamentals of autoencoders to include activation functions and optimizers. Chapter 3 describes the unique characteristics of our NAVFAC data and the manipulations

used to make the data set usable. It also provides a high-level comparison between the NAVFAC smart grid and the commercial smart grid. In Chapter 4, we provide an overview of the various test autoencoders built through the course of the research and identify the best performing model. We explain how an open-source Modbus data set was used on our autoencoders for benchmarking. Furthermore, we give an in-depth review of the best performing autoencoder and the comparison model described in [11], highlighting the differences between the two models. Chapter 5 describes and analyzes the autoencoder simulation results on the Modbus and NAVFAC data sets. We also show the comparative results between the developed autoencoder and the deep model given in [11]. Chapter 6 concludes the thesis and identifies future research opportunities.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Related Work

Researchers and academics have proposed several techniques to detect anomalous or malicious network traffic. This chapter addresses related work that focuses on the challenges inherent to network and power systems data sets and the different methods used for anomaly detection.

In [13], the author used a collection of Modbus datasets, produced by the University of Montreal using a supervisory control and data acquisition (SCADA) sandbox, similar to that of the Navy smart gird, to test the operability of supervised machine learning for anomaly detection in network activity. Using a Bayesian classification and machine learning algorithm, the author tested the model on seven different data sets, representing various attacks on the grid. The results showed that anomaly classification accuracy is affected by the size of the training data set and the amount of concentration of the malicious packets [13]. Furthermore, [5] used a k-nearest neighbor (KNN) machine learning approach to detect cyberattacks on a simulated network created by the Canadian Institute of Cybersecurity. The author showed that by optimizing the k-value, the KNN model could accurately identify and classify malicious activity. However, both researchers note the limitation of their data sets and the difficulty of acquiring labeled real-world data [5], [13].

To combat the data set labeling challenge, [11] introduced an unsupervised machine learning approach to anomaly detection using an IEEE 118-bus system dataset. The researchers developed an autoencoder to detect FDIAs using an hourly power load data set from 32 European countries. The researchers calculated "normal" power system states and corresponding measurements using optimal power flow solutions [11]. The authors demonstrated that the proposed autoencoder is superior to current anomaly detectors deployed on the power grid. With a detection probability in the mid to high 90%, the researchers showed that their proposed neural network does not need attack data for training. However, this data set consisted of macro-level power generation parameters (total power generation, total power consump-tion, installed capacity, price, etc.) rather than power analysis within a specific smart grid.

This thesis builds upon the research in [11] to address anomaly detection within a smart grid.

2.1 Background

2.1.1 Autoencoder Fundamentals

The model used in this research is an unsupervised machine learning algorithm called an autoencoder. An autoencoder is a neural network designed for representation learning by using a hidden layer (or "bottleneck") in the network, forcing a compressed knowledge representation of the original input [14].

To achieve representative learning, autoencoders apply backpropagation, setting the target values equal to the inputs. By defining the desired output as the input values, an autoencoder can train without using labeled data. This process performs dimensionality reduction by compressing each layer, in contrast to principal component analysis (PCA) which performs one massive transformation [15]. In [16], researchers demonstrated that autoencoders could detect subtle anomalies that linear PCA missed.

A basic autoencoder includes an encoder, bottleneck, and decoder as shown in Figure 2.1. The encoder compresses the input data into a latent space representation, reducing the dimensionality and distorting the original data. The bottleneck represents the compressed input, retaining only relevant information from the input [15]. The decoder decompresses the data back to the original dimension. The data is reconstructed from the latent space representation, which produces a lossy reconstruction of the original data [15].

To explain the learning process, consider an autoencoder with only an input and output layer. The learning process is defined by:

$$h = \sigma(W_{x_h} x + b_{x_h}) \tag{2.1}$$

$$z = \sigma(W_{h_x} h + b_{h_x}) \tag{2.2}$$

where σ is the nonlinear activation function, b is the bias, and W is the weight of the neural network [17].

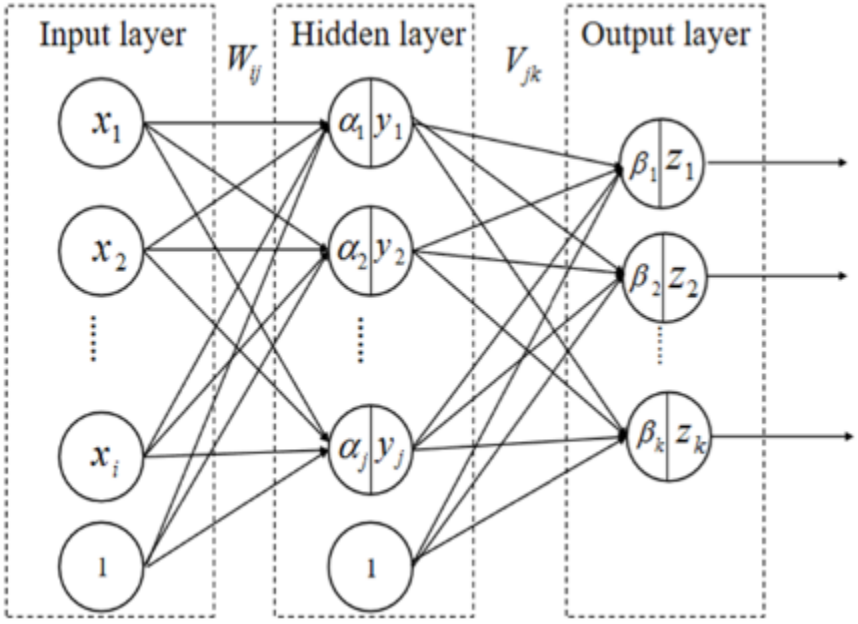

Figure 2.1. Single Layer Autoencoder Structure. Source: [16].

The autoencoder uses an activation function to transform the input vector x into a hidden representation h [17].

After the input vector is compressed into a hidden representation, we calculate the reconstruction error or loss function. For this research we use the mean squared error (MSE) regression model. The MSE is given by:

$$r = \frac{1}{N}\sum_{i=1}^{N}(x_i - z_i)^2 \tag{2.3}$$

where the difference between the reconstructed vector z (the model's prediction) and the original input vector x (ground truth) squared. The average is then taken across the entire training set.

To find the delta δ after calculating the reconstruction error, we use:

$$\delta = \sigma'(x_i) r_i \tag{2.4}$$

where σ' is the derivative of the activation function. For an autoencoder with multiple hidden layers, the delta is calculated at each layer and updated accordingly. This process continues until just before the input layer [17], [18]. The learning process updates parameters until the model converges [17].

We chose an autoencoder because of its ability to learn without labeled data and its simplicity. Autoencoders are easy to tune and experiment with, making them an excellent foundation for additional research on this topic. We tested various autoencoder structures for this research, experimenting with adding layers, varying compression amounts, and using different types of layers. The models and their performance are further discussed in Chapter 4 and 5, respectively.

2.1.2 Activation Functions: Sigmoid versus ReLU

Artificial neural networks (ANN) were originally designed to mimic biological neural networks [19]. Whereas biological neurons receive electrical signals, an ANN neuron receives weights and biases, as described in Section 2.1.1. After the input layer, the remaining layers use an activation function to transform the weighted sum of the inputs into an output. For our tests, the model used either sigmoid or rectified linear unit (ReLU) activation from the first to the penultimate layer. We wanted to test whether there was a significant performance enhancement using one or the other. The output layer uses the linear activation function because we want unbounded output values.

Sigmoid Activation

Sigmoid activation is a common activation function used for shallow ANNs. It is called a logistic function because it outputs values between zero and one. The sigmoid function is given by:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

and its derivative is defined by:

$$g'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{2.6}$$

as seen in Figure 2.2.

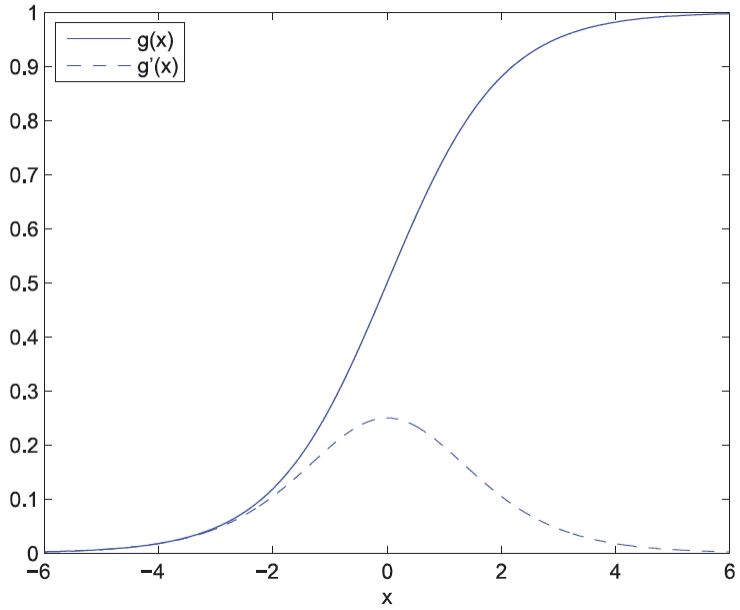

Figure 2.2. The Graphic Depiction of Sigmoid Function and Its Derivative. Source: [18].

Since the sigmoid function is a continuous function, it is differentiable everywhere, which is desirable. However, the literature indicates that sigmoid should not be used for deep ANN due to a saturation issue. Specifically, sigmoid produces zero gradient in the limit. The limit for sigmoid's derivative is defined by [19]:

$$\lim_{x \to +\infty} g'(x) = 0 \tag{2.7}$$

$$\lim_{x \to -\infty} g'(x) = 0 \tag{2.8}$$

as seen in Figure 2.2. This phenomenon is called the vanishing gradient, which restricts the contributions of the first several layers to the learning process during training. In response to this issue, researchers experimented with other activation designs [19].

ReLU Activation

One alternative to sigmoid is ReLU activation, which addresses the vanishing gradient problem. The ReLU function is given by:

$$g(x) = max(0, x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0, \end{cases} \tag{2.9}$$

and its derivative is defined by:

$$g'(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0, \end{cases} \tag{2.10}$$

as shown in Figure 2.3. Rather than restricting the output to between zero and one, ReLU activation output is either zero, if the input x is negative, or the output equals x, if x is positive [19]. Most academic literature favors the ReLU activation function because of its ability to converge quickly, overcome local optimization, and resolve the vanishing gradient descent problem [19].

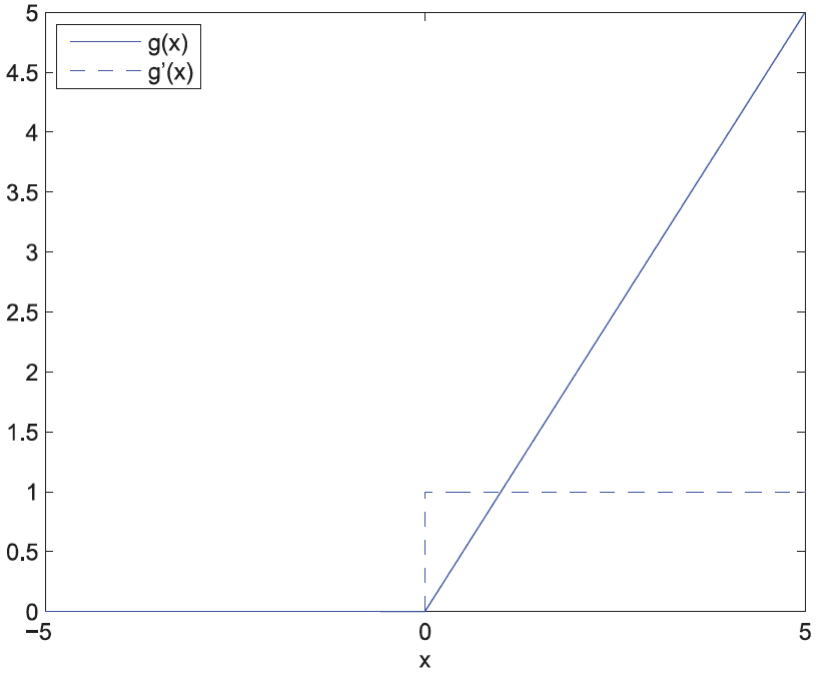

Figure 2.3. The Graphic Depiction of ReLU Function and Its Derivative. Source: [18].

2.2 Optimization Function: Stochastic Gradient Descent (SGD) versus Adaptive Moment Estimation (Adam)

After the activation function is applied to the last layer of the encoder, the values transfer to an optimizer. In machine learning, optimizers are combined with a backpropagation algorithm to minimize the cost of computing the gradient [20]. Optimization functions are designed to find the global minimum, the lowest point on a curve as depicted in Figure 2.4. We experimented with two optimizers – stochastic gradient descent (SGD) and adaptive moment estimation (Adam). With SGD, the network parameters are updated as the optimizer processes each mini-batch of training data as opposed to adjusting after the entire training data set has been evaluated. Adam is a follow-on to SGD that endeavors to increase efficiency. Adam gives each parameter its own learning rate that is adapted as the training progresses based on previous adjustments, enabling Adam to find the global minimum more quickly [21]. Literature indicates that Adam performs better during training but, in some

cases, generalizes worse than SGD [21].

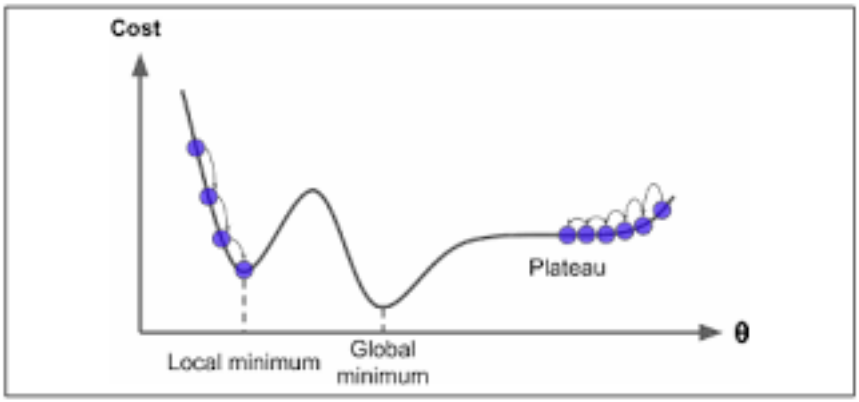

Figure 2.4. The Graphic Representation of Local and Global Minimum. Source: [22].

2.3 Chapter Summary

The literature has shown that autoencoders are efficient, powerful unsupervised machine learning models. Our research incorporates commonly used activation and optimization functions to increase generalizability while comparing performance across various combinations.

CHAPTER 3:
Dataset and Preprocessing

In this chapter, we discuss our methodology to process the NAVFAC AMI data set. We begin by providing a comparison of a commercial smart grid to the NAVFAC smart grid. After describing the grids, we detail the specifics of the NAVFAC data set and the manipulations used to convert the data into a usable format. We conclude the chapter by explaining how FDIA was used in this thesis.

3.1 NAVFAC Dataset

One of the key elements for smart grids is the AMI. The AMI consists of three sub-systems: data management system, communication network, and smart devices [22]. Figure 3.1 displays a generic view of this structure. We use Figure 3.1 to compare the commercial smart grid infrastructure to the NAVFAC smart grid system.

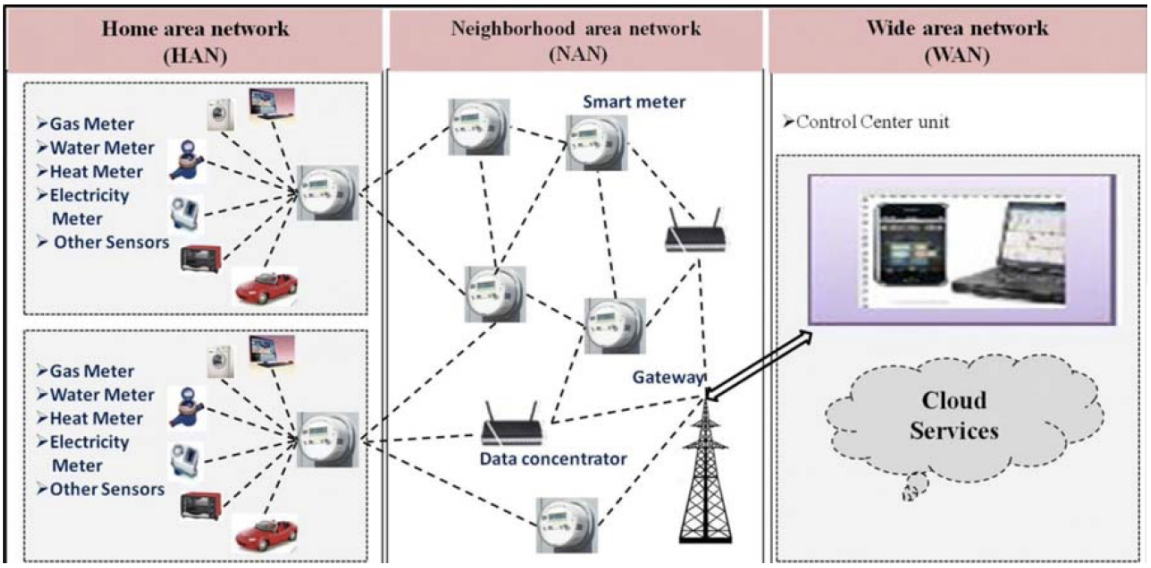

Figure 3.1. Multi-layer Smart Grid Communications Network. Source: [28].

Figure 3.2 displays the architecture and communication flow for the Navy smart grid provided by NAVFAC. At the bottom of the three ovals - Base X, Base Y, and Base Z –

is the AMI. The AMI meters and base computer application software (BCS), the software designed to facilitate two-way communication, record and communicate data collected at each site. This function corresponds to the home area network (HAN) shown in Figure 3.1. A HAN enables multiple devices within the home to communicate with the grid, helping to tailor power supply to meet the demand [23]. Navy smart meters could transmit data from buildings, ships, piers, or other critical assets.

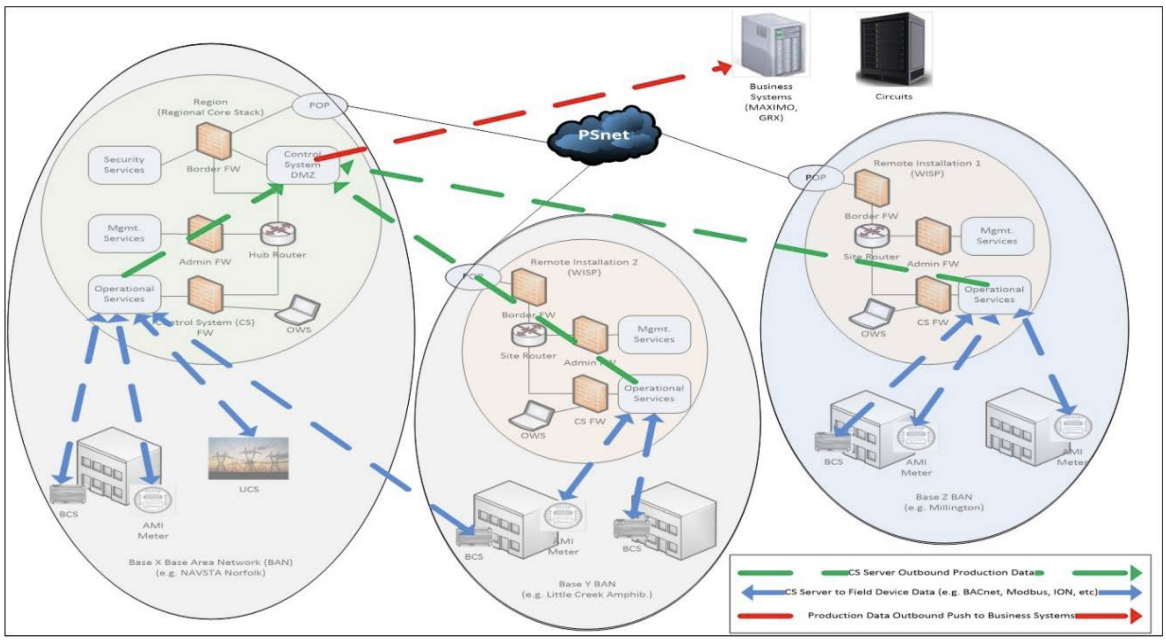

Figure 3.2. High Level View of the NAVFAC Smart Grid System Architecture. Data flow paths between various devices are also shown. Source: [2].

In the Navy smart grid, the AMI devices communicate to operational services. This exchange corresponds to data concentrators in the generic structure shown in Figure 3.1. These devices aggregate the data collected from each meter in a determined area. Commercial smart grids often refer to this region as neighborhood area networks (NAN), as depicted in Figure 3.1. The Navy smart grid combines the functions of the HAN and NAN into what it calls the base area network (BAN). The BAN provides a high-level view of communication flows and activity within the base's smart grid from the AMI to the edge device.

After the operational services receive the data, the information is transmitted to the control system demilitarized zone (DMZ). The DMZ attempts to filter out malicious traffic while

allowing legitimate traffic into the network. The DMZ is the boundary between the Navy smart grid and the Internet. This boundary has its own set of challenges that is beyond the scope of research presented in this thesis. The final route from the DMZ to the Internet aligns with the WAN displayed in Figure 3.1. The data set used in this research applies to information exchanged from the AMI to the operational services.

For this research, we used a NAVFAC AMI time series data set. The data set includes 35 metering statistics (power, current, voltage, phase, frequency, etc.) on 1,200+ AMI devices. The system recorded meter readings once an hour every day from the beginning of January 2020 to the end of January 2021, equating to 9,528 inputs per meter statistic. Table 3.1 displays a subset of the NAVFAC data input features used in our experiments. Specifically, the table displays 41 samples of the three-phase current (columns A, B, and C), calculated neutral current (column D), and frequency (column E) for one AMI meter. A full list of the data set input features is provided in Appendix A.

Table 3.1. Subset of NAVFAC Data and Input Features - (from left to right) three-phase current, neutral current, and frequency.

A	B	C	D	E
ML_CH_EL_BL_CAD108_1_AmpsA	ML_CH_EL_BL_CAD108_1_AmpsB	ML_CH_EL_BL_CAD108_1_AmpsC	ML_CH_EL_BL_CAD108_1_AmpsN	ML_CH_EL_BL_CAD108_1_Frequency
19.80223316	28.04912444	28.07564029	10.26442111	59.99874222
20.08867897	29.6083119	31.03246116	10.09458527	60.00093188
19.31412804	28.75936009	30.81739706	9.749406079	60.00312154
21.37585138	29.08793374	29.24320396	9.404226884	60.00531501
19.4365944	29.71736718	30.99198873	9.059047689	60.00750467
19.42904961	28.39457105	28.63253806	8.713868495	60.00969814
19.16327852	29.0129244	28.46709203	8.3686893	60.0118878
21.33285703	30.30423428	28.74542657	8.023510105	60.01407746
19.23081021	32.92819507	28.99151115	7.678330433	60.01589709
19.31927049	29.34479837	28.19540429	7.668867982	60.0155385
21.74381065	30.1420508	29.87955905	7.73688996	60.01517992
19.31481622	28.23499972	28.55614818	7.804912414	60.01482133
19.60037663	26.50845905	28.1557669	7.872934392	60.01446275
20.50251539	27.59935798	28.67150825	7.94095637	60.01410416
20.82830276	31.68289663	28.89007619	8.008978824	60.01374558
21.44134371	32.96617496	29.08904783	8.077001279	60.01338699
19.34298956	29.27197946	29.55821127	8.145022779	60.01316192
19.42616472	28.81698243	30.23903358	8.514430313	60.01370362
19.18879147	28.1380505	28.30517502	8.931898743	60.01424531
20.06196801	30.32682457	29.12753966	9.349367174	60.014787
20.66791946	33.76398128	35.28367612	9.766835605	60.01532869
20.71311053	33.40337797	29.82834641	10.18430404	60.01587039
22.35909867	33.8992188	30.81066844	10.60177247	60.01641208
22.24169084	34.98733565	29.58121488	11.0192409	60.01695377
22.6666724	35.10792001	30.97708379	11.43670838	60.01730473
21.30970864	34.80672452	29.55660831	11.41746304	60.01675922
21.19914693	34.18040467	30.12945798	11.38890306	60.01621753
20.55455048	33.3291806	35.28059191	11.36034309	60.01567583
20.36251098	32.54070664	30.55274006	11.33178217	60.01513414
21.94516993	31.84016215	30.36826213	11.3032222	60.01459245
19.92621435	32.71579766	30.35516996	11.27466223	60.01405076
27.36852663	36.63082285	32.45240582	11.24610226	60.01350906
40.99850568	37.87734868	38.45874182	11.20643187	60.01223113
39.09131613	41.26671477	36.00549959	10.79661294	60.00822184
27.8111973	40.94595236	36.9662594	10.38679401	60.00420874
49.17045477	37.31666383	35.29385193	9.976975074	60.00019945
33.18045227	33.51311932	35.28742334	9.567156141	59.99619017
31.20373573	36.13684093	35.51639367	9.157337208	59.99218088
40.96486048	34.53898121	36.20859611	8.747518276	59.98817159
20.80058051	29.2923017	33.14574752	8.337698389	59.98416231
19.17330536	30.29594257	28.24703025	8.037771493	59.98089308

To run our models, we used TensorFlow. TensorFlow is an end-to-end open-source machine learning platform designed to build and train machine learning models using high-level APIs. TensorFlow enabled us to quickly test and debug our models, resulting in rapid model design and final product development [24]. The scripts were written in Python for ease of use and portability. Code is provided in Appendix B through F.

3.2 Data Preprocessing

NAVFAC provided 13 time series data sets as .xml files. Each dataset represents the month the data was collected. The data was formatted such that the first column identified the

device and the meter statistic. The corresponding row values were the readings taken every hour that month.

To get our data in a usable format, we manipulated the NAVFAC spreadsheets. In the original spreadsheets, the first column represented the AMI device statistic and the top row corresponded to the timestamp data – the time the reading was recorded, as shown in Table 3.2. When using spreadsheets in Python, the columns are the input parameters. Since we wanted the AMI device readings to be the input parameters, we transposed the files. This manipulation allowed us to remove the timestamp, which was irrelevant for our experiments, and format each column to correspond to a meter statistic.

After dropping the time series data, we removed all input features that were not statistical values ('yes'/'no' entries). This reduction compressed our feature input space from 35 inputs to 26 per AMI. From there, we converted all input values that had error readings (such as 'I/O Timeout' or 'No Data') to the average of its corresponding column. We converted approximately 16% of the data due to error readings. Following the data manipulation, we concatenated the 13 .xml files into one file.

Table 3.2. Subset of the original NAVFAC data set with time series information across the top row and the AMI metering statistics in the first column.

A	B	C	D	E	F
Name	01-Jan-20 00:00:00	01-Jan-20 01:00:00	01-Jan-20 02:00:00	01-Jan-20 03:00:00	01-Jan-20 04:00:00
ML_CH_EL_BL_CAD108_1_AmpsA	18.00185013	18.26225281	17.55812263	19.43239784	17.66945457
ML_CH_EL_BL_CAD108_1_AmpsB	25.49894905	26.91637802	26.14461136	26.44331169	27.01551819
ML_CH_EL_BL_CAD108_1_AmpsC	25.52305412	28.21104622	28.01553535	26.58446503	28.17425346
ML_CH_EL_BL_CAD108_1_AmpsN	10.26431847	10.09448433	9.749308586	9.404132843	9.0589571
ML_CH_EL_BL_CAD108_1_BatteryStatus	0	0	0	0	0
ML_CH_EL_BL_CAD108_1_Frequency	59.99814224	60.00033188	60.00252151	60.00471497	60.0069046
ML_CH_EL_BL_CAD108_1_INTERVALKVARDEMAND	1.317835093	1.325741768	1.334745765	1.343749762	1.352753639
ML_CH_EL_BL_CAD108_1_INTERVALKWDEMAND	8.39419651	8.390296936	8.386398315	8.382499695	8.378601074
ML_CH_EL_BL_CAD108_1_KVAR	1.207982063	1.220406771	1.244156957	1.267907023	1.29165709
ML_CH_EL_BL_CAD108_1_KVARHD	0.335000008	0.335000008	0.335000008	0.335000008	0.335000008
ML_CH_EL_BL_CAD108_1_KVARHR	0	0	0	0	0
ML_CH_EL_BL_CAD108_1_KW	8.149680138	8.18230629	8.214933395	8.247559547	8.280185699
ML_CH_EL_BL_CAD108_1_KWHD	2.086999893	2.086999893	2.086999893	2.086999893	2.086999893
ML_CH_EL_BL_CAD108_1_KWHDTOTAL	383700.6715	383709.1141	383717.5567	383725.9993	383734.4418
ML_CH_EL_BL_CAD108_1_KWHR	0	0	0	0	0
ML_CH_EL_BL_CAD108_1_KWHRTOTAL	0	0	0	0	0
ML_CH_EL_BL_CAD108_1_MAXKVARDEMAND	2.327378035	2.327378035	2.327378035	2.327378035	2.327378035
ML_CH_EL_BL_CAD108_1_MAXKWDEMAND	19.34158897	19.34158897	19.34158897	19.34158897	19.34158897
ML_CH_EL_BL_CAD108_1_PhAB_Voltage_PA	29.87999916	29.87999916	29.87999916	29.87999916	29.87999916
ML_CH_EL_BL_CAD108_1_PhA_Current_PA	5.070000172	6.449999809	4.150000095	5.190000057	12.31000042
ML_CH_EL_BL_CAD108_1_PhA_Voltage_PA	0	0	0	0	0
ML_CH_EL_BL_CAD108_1_PhBC_Voltage_PA	-89.73000336	-89.73000336	-89.73000336	-89.73000336	-89.73000336
ML_CH_EL_BL_CAD108_1_PhB_Current_PA	-107.9700012	-106.6100006	-106.7200012	-107.4100037	-107.6399994
ML_CH_EL_BL_CAD108_1_PhB_Voltage_PA	-119.5899963	-119.5899963	-119.5899963	-119.5899963	-119.5899963
ML_CH_EL_BL_CAD108_1_PhCA_Voltage_PA	150.3500061	150.3500061	150.3500061	150.3500061	150.3500061
ML_CH_EL_BL_CAD108_1_PhC_Current_PA	125.9199982	128.8000031	126.1699982	127.3799973	126.3700027
ML_CH_EL_BL_CAD108_1_PhC_Voltage_PA	120.3199997	120.3199997	120.3199997	120.3199997	120.3199997
ML_CH_EL_BL_CAD108_1_PowerFactor	0.98299998	0.98299998	0.98299998	0.987999976	0.987999976
ML_CH_EL_BL_CAD108_1_THDA	10.94	10.94	10.94	10.38	10.38
ML_CH_EL_BL_CAD108_1_VoltsA	121.7668457	121.7091293	121.6483688	121.5876007	121.5268402
ML_CH_EL_BL_CAD108_1_VoltsAB	210.5512543	210.4061584	210.2513733	210.0965881	209.9418182
ML_CH_EL_BL_CAD108_1_VoltsB	120.9006348	120.8249588	120.7286377	120.632309	120.5359802
ML_CH_EL_BL_CAD108_1_VoltsBC	209.3505859	209.1885529	209.0265045	208.8644714	208.7024384
ML_CH_EL_BL_CAD108_1_VoltsC	121.0372696	121.0324554	121.0365982	121.040741	121.0448914
ML_CH_EL_BL_CAD108_1_VoltsCA	211.0124054	210.8993988	210.7797852	210.6601563	210.5405273
ML_CH_EL_BL_CAD109_1_AmpsA	30.19542694	27.83498955	24.05105782	28.16897202	39.9201622
ML_CH_EL_BL_CAD109_1_AmpsB	9.992212296	11.78724957	10.60418606	10.05705929	10.78899765
ML_CH_EL_BL_CAD109_1_AmpsC	22.64203644	18.66898727	6.604632854	24.64419937	19.07324409
ML_CH_EL_BL_CAD109_1_AmpsN	15.75883198	14.8468256	13.49901581	12.15120697	10.80339718
ML_CH_EL_BL_CAD109_1_BatteryStatus	0	0	0	0	0
ML_CH_EL_BL_CAD109_1_Frequency	59.99948502	59.99857712	59.99766541	59.99675751	59.99584579
ML_CH_EL_BL_CAD109_1_INTERVALKVARDEMAND	-0.050591383	-0.044580456	0.005211	0.055002455	0.104793906

Once the dataset was in one file, our last step was the insertion of an FDIA. To simulate an FDIA, we increased 10% of the data by a certain percentage, mimicking the research performed in [11]. For this research, we performed two types of FDIA. The first attack selected all input values and increased them from 1%-10%, incrementing by one. The range started at 1% to find the smallest percent increase that was detectable with acceptable results. It ended at 10% because our experiments showed that the performance plateaued at this upper threshold value. The results of these experiments will be discussed in further detail in Section 5.3. The second attack increased all voltage, current, and power values using the

power equation defined by:

$$P = VI \tag{3.1}$$

where V is voltage and I is current. This attack changed roughly 40% of the input features. The voltage and current values were increased from 2%-5%, incrementing by one, with the power values corresponding to the equation. The percent increase for the second attack started at 2% because a 1% increase would have negligible effect on the power values. Furthermore, the percent increase for voltage and current stopped at 5% because the corresponding power change in that scenario is a 25% increase. We determined that a power increase above 25% would no longer constitute a nuanced FDIA.

To separate the data between normal and anomalous activity, we added a 'Malicious' column to the spreadsheet. For this column, we define 1 and 0 to represent 'Malicious' and 'Normal', respectively. Therefore, the ten percent of data whose values were increased to mimic an FDIA, a 1 was placed in the 'Malicious" column. The remainder of the data received a 0 in that column.

3.3 Chapter Summary

Manipulating data into a usable form is critical to machine learning. The NAVFAC data set required removing the time series information, eliminating non-numeric values, and increasing certain portions of the data to mimic an FDIA. The next chapter will describe the models used to process this data.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Proposed Autoencoder Models

To establish a baseline for this research, several autoencoders were built and tested. Before training and testing the models on the NAVFAC data set, the autoencoders were tested on an open-source Modbus data set to determine the best performing model. Performance was determined by model accuracy, recall, and precision scores. Once the best performing model was identified, we trained the model on the NAVFAC AMI data set. We compared the performance of our model to the autoencoder presented in [11]. The autoencoder in [11] was also trained on the NAVFAC AMI data set for accurate comparison. We chose the autoencoder in [11] because it proved effective at detecting anomalous activity on a traditional power grid data set and its structure is deeper than the models generated in this research.

The remainder of this chapter discusses the open-source Modbus data set initially used for benchmarking. This is followed by discussion on the various test autoencoders built and a description of the best performing model. We also provide an overview of the autoencoder presented in [11].

4.1 Modbus Data Set

Researchers from the University of Montreal generated a Modbus data set via a SCADA sandbox, using electrical network simulators [13]. The system design used a master terminal unit (MTU), which was responsible for maintaining information about the state of the physical process, and remote terminal units (RTU), which controlled the communication with the MTU [13]. The sandbox communication scheme used the Modbus protocol, a popular SCADA protocol similar to the one used in the NAVFAC smart grid [4]. Modbus performs continuous polling of each smart meter, relaying that information from the RTU to the MTU. The information exchanged can be viewed and interpreted by a human operator [13].

The Modbus data includes eleven separate SCADA data sets. Six data sets contain only normal network traffic, and five data sets have a mix of legitimate and malicious network

traffic. From these data sets, our research selected four mixed data sets. These four data sets are described in the first four rows in Table 4.1. Additionally, for the purpose of our research presented in this thesis, we created another data set that combined all the data sets with mixed network traffic. This is denoted as the Combined Data Set and is shown in the last row of Table 4.1.

Table 4.1. Description of Modbus Data Sets. The first four rows of data sets were provided by the researchers in [30]. The Combined Data Set, shown in the last row, was compiled specially for the research in this thesis. Source: [30].

Name	Description	Malicious Activity and Percent	Number of entries
Moving_two_files_Modbus_6RTU	3 minutes of regular Modbus traffic including polling only – 1 MTU, 6 RTU, and 10 seconds polling interval	Yes: 2.2%	3319
Send_a_fake_command_Modbus_6RTU_with_operate	11 minutes of regular Modbus traffic, including polling and manual operation – 1 MTU, 6 RTU, and 10 seconds polling interval. It also includes sending Modbus write operation from a compromised RTU using Metasploit proxy functionality and the proxychains tool	Yes: 0.09%	11,166
CnC_uploading_exe_modbus_6RTU_with_operate	1 minute of regular Modbus traffic, including polling and manual operation – 1 MTU, 6 RTU, and 10 seconds polling interval. It also includes sending an EXE file from a compromised RTU through a Metasploit meterpreter channel	Yes: 8%	1,426
6RTU_with_operate	5 minutes of regular Modbus traffic, including polling and manual operation – 1 MTU, 6 RTU, and 10 seconds polling interval. It also includes using an exploit (ms08_netapi) from a compromised RTU to compromise another RTU using Metasploit	Yes: 64%	1,856
Combined Data Set	Amalgamation of all the malicious activity generated by the other data sets.	Yes: 7%	17,767

4.2 Test Autoencoders

To determine the best performing model (trained on the Modbus data set described in the previous section), our research experimented with several different structures. This section discusses their general design and the thought-process behind the attempts. The results for each model are discussed in Chapter 5.

Our first experiment was a two-layer autoencoder that compressed the input features by half

at the first layer and then to two nodes at the second layer, as shown in Figure 4.1. The reasoning behind this structure was to force the autoencoder to make a binary decision – either the input was normal or malicious. However, this autoencoder produced poor results. We refer to this model as Test Autoencoder Model 1.

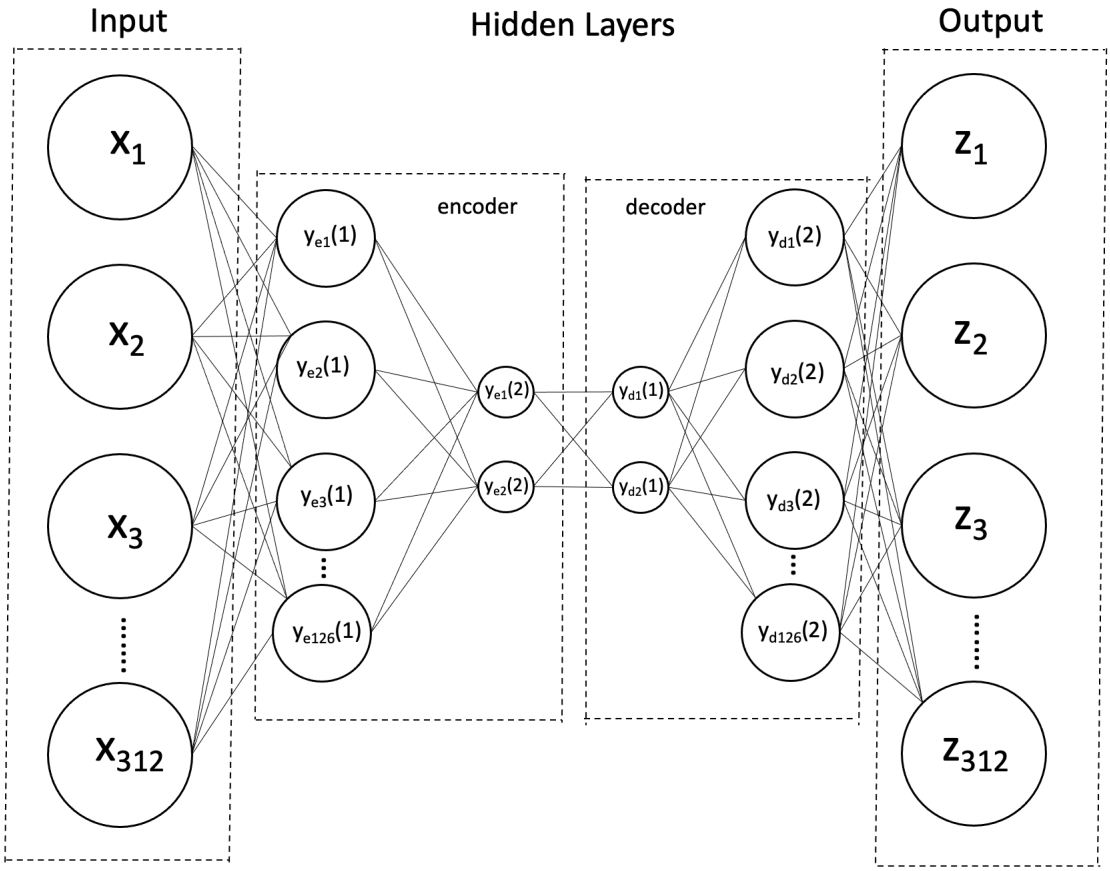

Figure 4.1. Test Autoencoder Model 1.

For the second autoencoder, another layer was added. The model compressed the inputs by half, then to ten nodes, and then to two nodes as depicted in Figure 4.2. The rationale for the additional layer was to expand the depth of the model, increasing the number of opportunities to learn the latent space representation while maintaining a binary bottleneck. However, the model failed to improve. We refer to this model as Test Autoencoder Model 2.

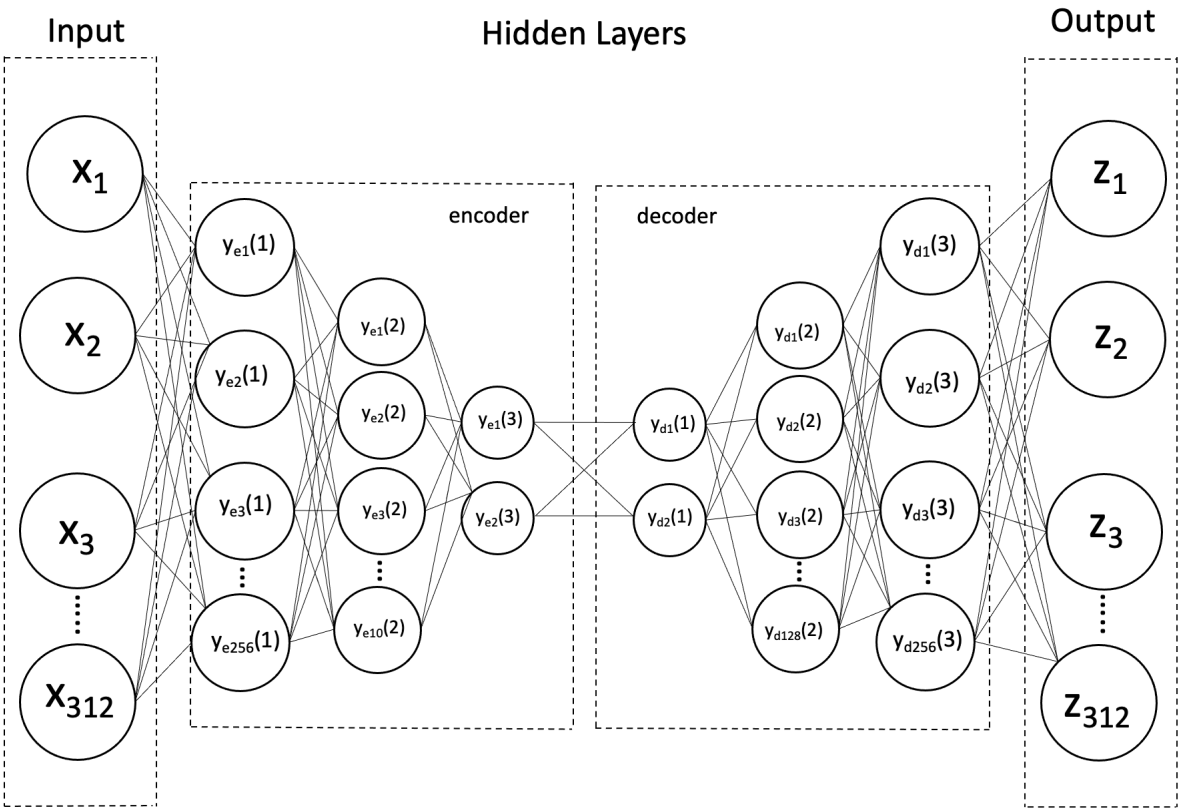

Figure 4.2. Test Autoencoder Model 2.

Since the Test Autoencoder Model 2 did not improve from the Test Autoencoder Model 1, we reverted to a two-layer structure. The third model (referred to as Test Autoencoder Model 3) kept the same first layer but changed the compression at the bottleneck. Rather than two nodes, like the previous two models, the second layer was compressed to five nodes, as shown in Figure 4.3. This change challenged the assumption that the best autoencoder needed to be compressed to a binary decision space.

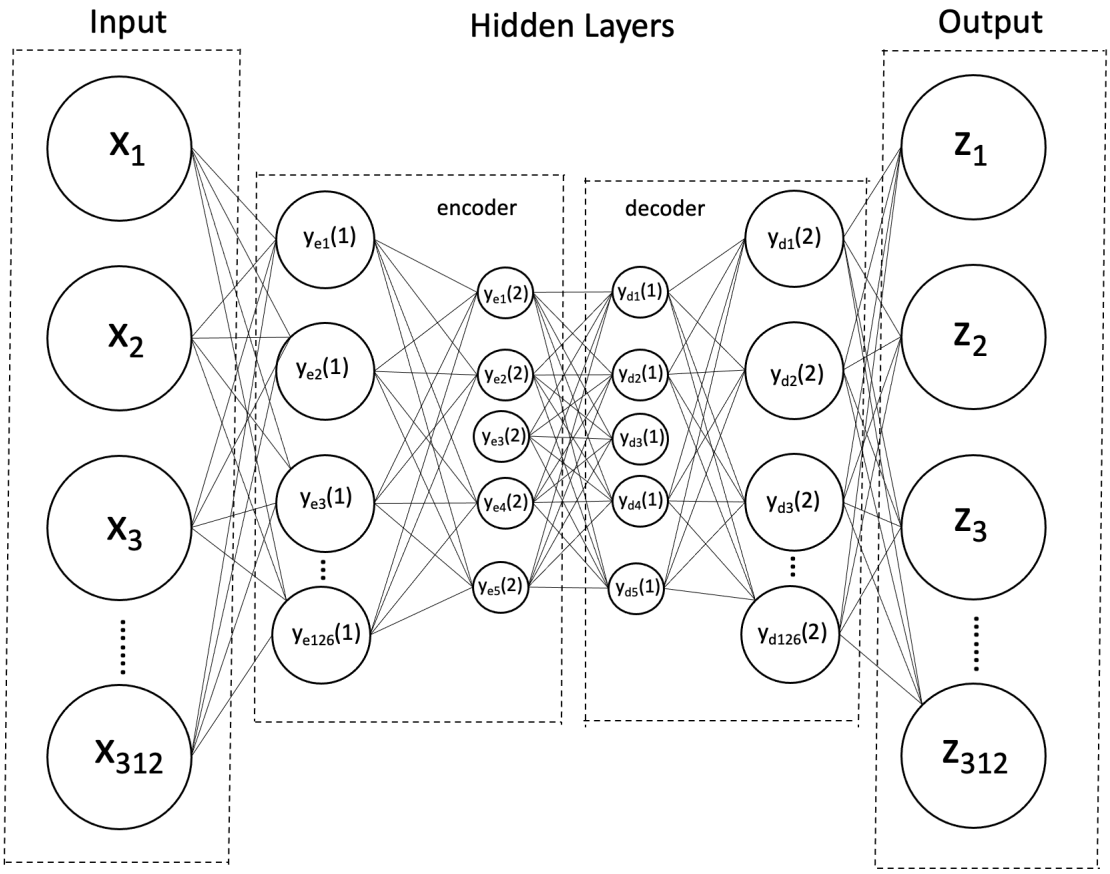

Figure 4.3. Test Autoencoder Model 3.

The Test Autoencoder Model 3 performed significantly better than the previous models. From this point, we ran two more tests - Test Autoencoder Models 4 and 5. The first layer remained the same, and the second layer was compressed to four nodes for the Test Autoencoder Model 4 and to three nodes for Test Autoencoder Model 5. The performance across Test Models 3, 4, and 5 were similar. However, after numerous tests, Test Autoencoder Model 4 consistently produced slightly better accuracy, recall, and precision scores. For the remainder of this thesis, we refer to this best performing model as the Optimal Model. The Optimal Model is discussed in more detail in the next section.

4.3 Optimal Model

The Optimal Model includes two dense layers of compression and then the reconstruction. The input layer is reduced by half and then compressed to four nodes. From there, the model expands using the same dimensions as the encoding layer back to its original size. Figure 4.4 provides a structural diagram of the Optimal Model.

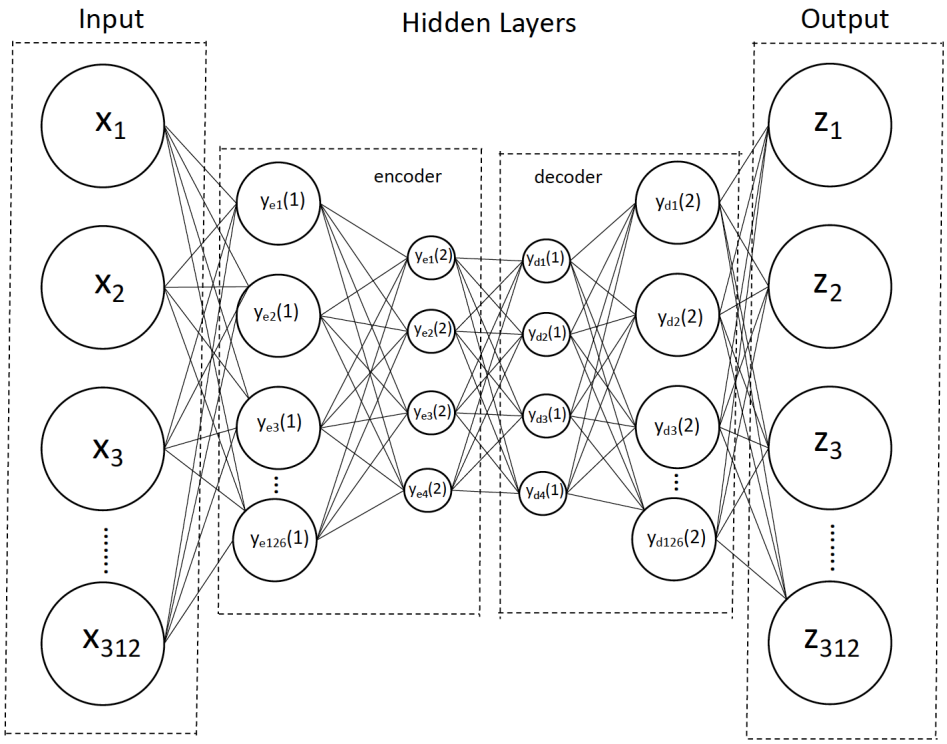

Figure 4.4. Optimal Autoencoder Structure - Four Hidden Layers.

The model used either ReLU or sigmoid activation from the first to the penultimate layer. We wanted to test whether there was a significant performance enhancement using one or the other. For ReLU activation, the output is either zero, if the input x is negative, or the output equals x, if x is positive. Sigmoid activation restricts the output between zero and one [19]. Most academic literature favors the ReLU activation function because of its ability to converge quickly, overcome local optimization, and resolve the vanishing gradient descent problem [19]. The output layer uses the linear activation function because we want unbounded output values.

We also experimented with two optimizers – Adam and SGD. Optimizers are designed to reduce the overall loss function and improve accuracy by modifying the weights and learning rate. Adam gives each parameter its own learning rate that is adapted as the training progresses. Conversely, SGD uses a scalar learning rate uniformly for all parameters [21].

Moreover, in TensorFlow, our models used a batch size of 128 and a learning rate of 0.01 for the Adam optimizer or 0.001 for the SGD optimizer. The model ran for between 700 and 1,500 epochs depending on the data set. We used the 'early stop' callback function to prevent overfitting.

We must note that this model achieves optimality in terms of precision, recall, and accuracy for the data sets specifically used in this thesis. Model performance may change with different data sets, and therefore we do not extrapolate our definition of optimality to other data sets.

4.4 Deep Model

The Deep Model autoencoder was developed in [11] and is used for comparison against the Optimal Model discussed in Section 4.3. The Deep Model autoencoder includes four dense layers of compression and decompression, as described in [11]. Using the powers of two, the compression begins at 2^8 (256) and decrements the exponent value by one at each layer until it reaches 2^5 (32). The decompression phase uses the inverse of this process. Figure 4.5 illustrates the architecture of the Deep Model.

Like the Optimal Model, the Deep Model used either sigmoid or ReLU activation from the first to the penultimate layer. Similarly, the output layer used the linear activation function because we wanted to compare its output values with the Optimal Model. The Deep Model used the same batch size and learning rates for each of the optimizers as the Optimal Model. The model ran for a similar number of epochs as the Optimal Model, but it took two to five times longer to converge. The time it takes a model to converge is an important consideration in a real time system. Since it has more layers and compression, the Deep Model will always run slower than the Optimal Model, despite all other parameters being equal.

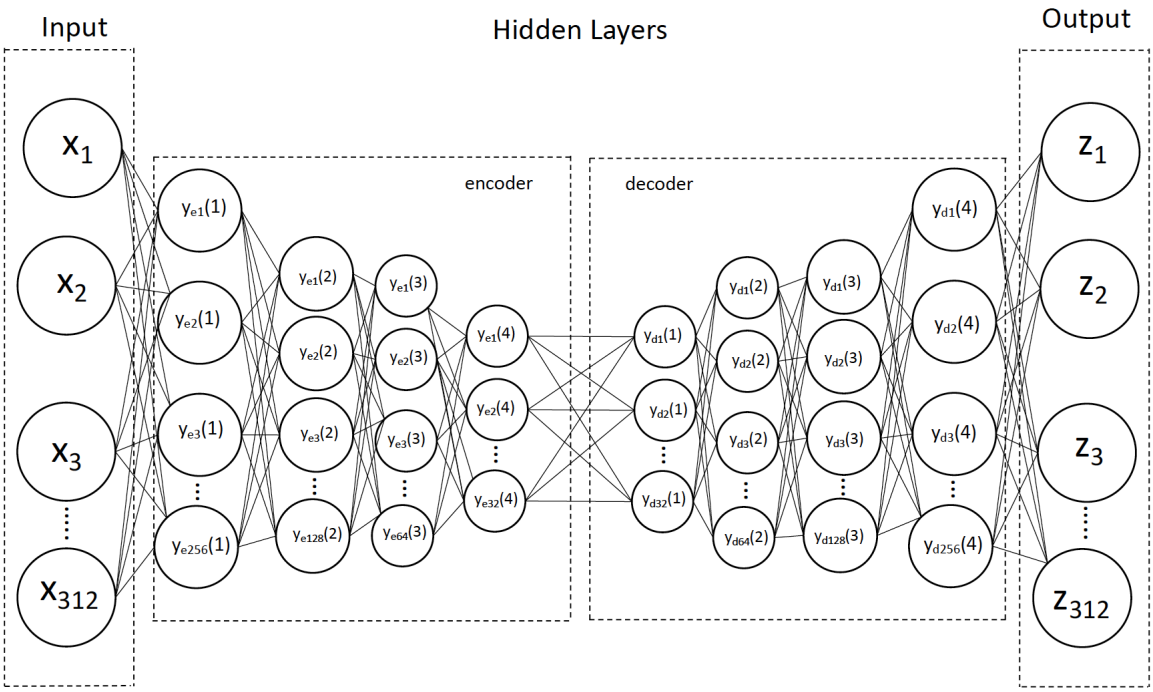

Figure 4.5. Deep Autoencoder Structure - Eight Hidden Layers.

4.5 Model Operation

To start the machine learning process, we segmented the data into two sets. One data set contains only normal activity data for training, and the other includes both normal and malicious activity data for testing. This process is shown in Figure 4.6. Once the training data is input into the model, the autoencoder tries to replicate the input values as its output, minimizing the reconstruction error, as described in Chapter 3. The model will continue training until the model converges or the last epoch runs.

To perform anomaly detection with an autoencoder, we must calculate a threshold. To determine the threshold, we feed only normal activity data into the model. For this research, we used the MSE as the loss function to calculate the reconstruction error.

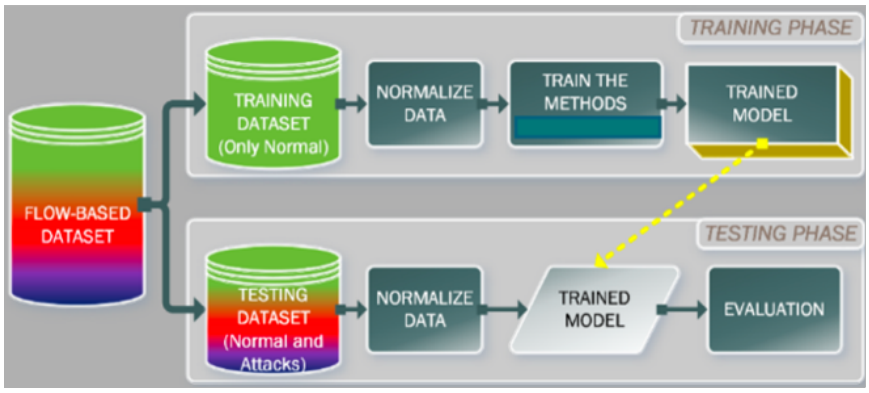

Figure 4.6. The Unsupervised Machine Learning Training and Testing Data Flow Process. Source: [15].

After the model calculates the MSE, the scores are rank-ordered. Next, we determined whether the MSE values are two standard deviations or greater from the average. If so, we removed those values. The process of removing the tail values in the distribution curve accounts for any strange but still normal data that entered the model. If we used the highest reconstruction error without removing the outliers, we might set an artificially high threshold, degrading model performance.

Once the tail values are removed, the MSE scores within two standard deviations are rank-ordered with the maximum value (the highest reconstruction error within the set) established as the threshold [25]. The autoencoder labels the input values with reconstruction errors above the threshold as 'malicious' and values below the threshold as 'normal.'

4.6 Chapter Summary

We built multiple models and identified the best performer, the Optimal Model, using a Modbus data set. The Optimal Model consists of four hidden layers. We also introduced a Deep Model discussed in [11] that we will use as a comparative performance benchmark. The Optimal Model and Deep Model will be trained on the NAVFAC data set. The results are discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results and Analysis

This chapter reviews the results of the experiments we conducted on the Modbus data set and the NAVFAC data set. The chapter begins by discussing the metrics used to determine the performance of our models. To help explain the metrics, we provide two graphical representations to visualize performance on a specific data set. Next, the test model results on the Modbus data set are discussed, highlighting the best performing autoencoder - the Optimal Model. The chapter concludes by comparing the results of the Optimal Model and the Deep Model on the NAVFAC data set and providing recommendations for employment in the Naval smart grid.

5.1 Data Analysis

Given that our autoencoder detects anomalous activity, the metrics of concern are accuracy and recall while maintaining an acceptable precision. Accuracy is the ratio of correctly labeled events to the total number of events. Recall is given by:

$$Recall = \frac{tp}{tp + fn} \tag{5.1}$$

where the ratio of the total true positive values (tp) predicted to the sum of true positive values and the total number of false negative numbers (fn).

Precision is defined by:

$$Precision = \frac{tp}{tp + fp} \tag{5.2}$$

where the ratio of the total true positives values (tp) predicted to the sum of true positive values and the total number of false positive values (fp) [26].

For an IDS, operators prefer an increased number of false positives (normal activity labeled

malicious) over false negatives (malicious data labeled normal). However, the model should not overtax an operator or system with excessive false alarms.

To help visualize the autoencoder performance and illustrate how to calculate recall and precision, two figures are provided. Figure 5.1 displays plotted predictions for one NAVFAC data set using the Optimal Model. The red line (threshold) represents the MSE after training on normal data only. The line divides the graph into two sections. Any error values above the line is predicted 'malicious', while everything below the line is predicted 'normal.' Orange and blue circles represent malicious and normal data, respectively. Blue circles plotted above the threshold are false positives, while orange circles below the threshold are false negatives. As shown, the model performed extremely well. It identified most of the anomalous activity, producing a small number of false negatives, without triggering too many false positives. The confusion matrix in Figure 5.2 shows the total number of true positives (top left), false positives (top right), false negatives (bottom left), and true negatives (bottom right).

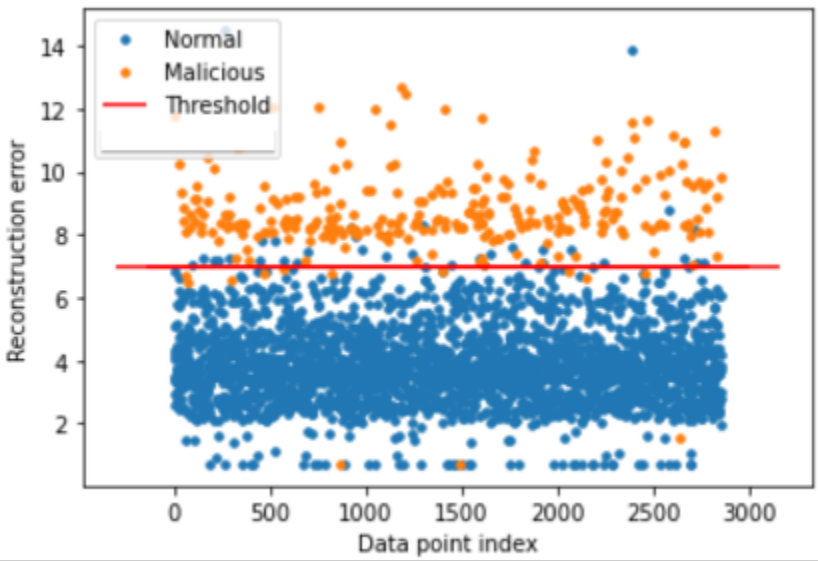

Figure 5.1. The Graphic Depiction of the Optimal Model Separating Data and Plotting Predictions on one NAVFAC Data Set.

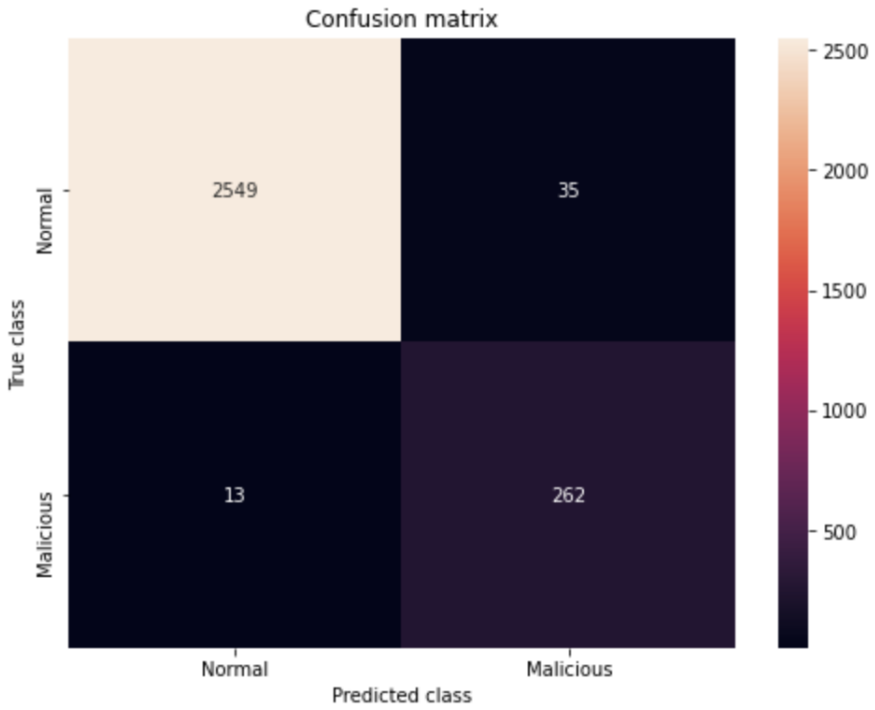

Figure 5.2. Confusion Matrix of the Optimal Model trained on one NAVFAC Data Set.

5.2 Modbus Model Results

This section describes the performance of the various test autoencoders that were discussed in Chapter 4. The results represent the performance of the test models trained on the Modbus data set [13]. The results for Test Autoencoder Model 1 (refer to Figure 4.1 for model structure) and Test Autoencoder Model 2 (refer to Figure 4.2 for model structure) are shown in Table 5.1 and Table 5.2, respectively. These models were by far the worst-performing autoencoders. Neither model achieved a 90% or better across accuracy, recall, and precision for any data set. Furthermore, they performed exceptionally poorly on the two most critical data sets - "6RTU with operation" and "Combined data" (as shown in Table 4.1). The "6RTU with operation" is the only data set with 50+% of the data labeled malicious. The "Combined data" represents the entirety of all normal and malicious data from the Modbus data set. This data set is most emblematic of a real-world environment.

The performance of Test Models 3 (refer to Figure 4.3), 4 (refer to Figure 4.4), and 5 improved significantly. Their results are shown in Table 5.3, Table 5.4, and Table 5.5, respectively. All three models scored mid to high 90% on the three metrics for multiple data sets. The only data set that the three models performed poorly on was "Send a fake command Modbus RTU." This data set is highly unequal; only 0.09% of the 11,000+ data points were malicious. Due to this imbalance, the autoencoders were heavily penalized for a few mistakes, degrading results considerably. However, all three models performed exceptionally well on the two most important data sets - "6RTU with operation" and "Combined data". Although Test Autoencoder Models 3, 4, and 5 performed similarly, Test Autoencoder Model 4 scored slightly better overall. Therefore, Test Autoencoder Model 4 became the Optimal Model for this thesis.

Table 5.1. Test Autoencoder Model 1 Results on Modbus Data Set. Autoencoder failed to balance accuracy, recall, and precision for any data set.

Data Set	Accuracy	Recall	Precision
Moving two files Modbus 6RTU	98.8%	54.5%	100%
Send a fake command Modbus RTU	95.3%	0%	0%
CnC uploading exe Modbus 6RTU	97.6%	78.7%	100%
6RTU with operation	56.5%	32%	100%
Combined data	91%	2%	11.5%

Table 5.2. Test Autoencoder Model 2 Results on Modbus Data Set. Autoencoder failed to balance accuracy, recall, and precision for any data set.

Data Set	Accuracy	Recall	Precision
Moving two files Modbus 6RTU	93.7%	100%	26.1%
Send a fake command Modbus RTU	99.6%	100%	7.6%
CnC uploading exe Modbus 6RTU	97.6%	78.7%	100%
6RTU with operation	54.3%	32%	90.4%
Combined data	89.2%	3.6%	7.9%

Table 5.3. Test Autoencoder Model 3 Results on Modbus Data Set. Autoencoder achieved mid 90% performance across accuracy, recall, and precision on three of the five data sets.

Data Set	Accuracy	Recall	Precision
Moving two files Modbus 6RTU	99.8%	100%	95.6%
Send a fake command Modbus RTU	98.6%	100%	2.1%
CnC uploading exe Modbus 6RTU	97.6%	78.7%	100%
6RTU with operation	97.4%	97.7%	98.3%
Combined data	99.3%	98.3%	93.3%

Table 5.4. Test Autoencoder Model 4 Results on Modbus Data Set. Autoencoder achieved mid 90% performance across accuracy, recall, and precision on four of the five data sets, including a perfect score on the 'CnC uploading exe Modbus 6RTU' data set.

Data Set	Accuracy	Recall	Precision
Moving two files Modbus 6RTU	99.8%	95.4%	100%
Send a fake command Modbus RTU	99.7%	100%	12.5%
CnC uploading exe Modbus 6RTU	100%	100%	100%
6RTU with operation	99.1%	100%	98.6%
Combined data	99.1%	94.6%	94.9%

Table 5.5. Test Autoencoder Model 5 Results on Modbus Data Set. Autoencoder achieved mid 90% performance across accuracy, recall, and precision on two of the five data sets.

Data Set	Accuracy	Recall	Precision
Moving two files Modbus 6RTU	99.1%	95.4%	75%
Send a fake command Modbus RTU	98.3%	100%	1.8%
CnC uploading exe Modbus 6RTU	97.6%	78.7%	100%
6RTU with operation	98.7%	100%	98%
Combined data	99.1%	95.6%	94%

5.3 NAVFAC Data Set Results

For this research, we created 14 different data sets from the given NAVFAC data set. Ten of the data sets mimic an FDIA across all AMI input parameters, representing a broad attack on the smart grid. As discussed in Chapter 3, these attacks increase the input values across the AMI data set from 1%-10%, incrementing by one. In Table 5.6, Table 5.7, Table 5.8, and Table 5.9, these attacks correspond to the rows with only a single percent in the '% Increase'

column (1%-10%). The other four datasets simulate an FDIA on specific input parameters – voltage, current, and power. These changes to specific input parameters mimic nuanced attacks. These attacks are displayed in the rows with two distinct percentage values in the '% Increase' column (bottom 4 rows of Tables 5.6-5.9). The first percent value represents the percentage increase to the voltage and current parameters. The second percent value corresponds to the change in the power input parameter using the power equation discussed in Chapter 3.

5.3.1 Optimal Model - Sigmoid and Adam/SGD

Table 5.6 displays the Optimal Model's performance using the sigmoid activation function and the Adam or SGD optimizers. When testing against an FDIA across all AMI input parameters, a 3% increase and above produced impressive results, with only recall failing to score in the mid 90%. However, at the 4% mark and above, accuracy, recall, and precision were all in the mid to high 90%. When simulating an FDIA on specific AMI parameters, current and voltage, the model did not produce positive results until a 4% increase to the voltage and current parameters. At the 2-3% increase for current and voltage, the recall score ranges from mid 50% to high 60%. However, at the 4% mark, the Optimal Model produces high accuracy and precision, and acceptable recall. With a 5% increase, the model produced mid-to-high 90% across all these statistics.

Table 5.6. Results for Optimal Model using Sigmoid Activation Function and either Adam or SGD Optimization on NAVFAC Data Sets.

Sigmoid							
Adam				SGD			
% Increase	Accuracy	Recall	Precision	% Increase	Accuracy	Recall	Precision
10%	99.6%	1.0%	96.6%	10%	99.6%	1.0%	96.6%
9%	99.6%	1.0%	96.6%	9%	99.6%	1.0%	96.6%
8%	99.6%	1.0%	99.6%	8%	99.6%	1.0%	96.6%
7%	99.6%	99.7%	96.6%	7%	99.6%	99.7%	96.6%
6%	99.6%	99.7%	96.6%	6%	99.6%	99.7%	96.6%
5%	99.6%	99.5%	96.6%	5%	99.6%	99.5%	96.6%
4%	99.4%	97.6%	96.5%	4%	99.4%	97.4%	96.5%
3%	98.9%	92.9%	96.4%	3%	98.9%	92.6%	96.4%
2%	96.6%	69.2%	95.2%	2%	96.8%	70.7%	95.3%
1%	94.8%	50.0%	93.6%	1%	94.7%	49.8%	93.5%
VI 2% / P 4%	95.1%	54.1%	94.0%	VI 2% / P 4%	95.3%	55.6%	94.1%
VI 3% / P 9%	96.5%	67.7%	95.1%	VI 3% / P 9%	96.5%	68.3%	95.2%
VI 4% / P 16%	98.1%	84.0%	96.0%	VI 4% / P 16%	98.1%	84.0%	96.0%
VI 5% / P 25%	99%	94.2%	96.4%	VI 5% / P 25%	99.0%	93.3%	96.4%

5.3.2 Deep Model - Sigmoid and Adam/SGD

Table 5.7 shows the Deep Model performance using sigmoid activation and the Adam or SGD optimizers. Similar to the Optimal Model, at the 3% increase mark the Deep Model produced 98.9% accuracy, 92.9% recall, and 96.4% precision. Additionally, at 4% and above, accuracy, recall, and precision were all in the mid to high 90%. When simulating an FDIA on specific AMI parameters, the Deep Model mimicked the Optimal Model, failing to achieve positive results until a 4% increase to both parameters. At that point, accuracy and precision score in the mid to high 90%, and recall scores 84%. At the 5% increase mark, the Deep Model produced mid to high 90% accuracy, recall, and precision.

Table 5.7. Results for Deep Model using Sigmoid Activation Function and either Adam or SGD Optimization on NAVFAC Data Sets.

Sigmoid							
Adam				SGD			
% Increase	Accuracy	Recall	Precision	% Increase	Accuracy	Recall	Precision
10%	99.6%	1.0%	96.6%	10%	99.6%	1.0%	96.6%
9%	99.6%	1.0%	96.6%	9%	99.6%	1.0%	96.6%
8%	99.6%	1.0%	99.6%	8%	99.6%	1.0%	96.6%
7%	99.6%	99.7%	96.6%	7%	99.6%	99.7%	96.6%
6%	99.6%	99.7%	96.6%	6%	99.6%	99.7%	96.6%
5%	99.6%	99.5%	96.6%	5%	99.6%	99.5%	96.6%
4%	99.4%	97.8%	96.6%	4%	99.4%	97.4%	96.5%
3%	98.9%	92.9%	96.4%	3%	98.9%	92.6%	96.4%
2%	96.6%	69.4%	95.2%	2%	96.7%	70.5%	95.3%
1%	94.6%	48.3%	93.3%	1%	94.8%	50.7%	93.6%
VI 2% / P 4%	95.2%	54.8%	94.0%	VI 2% / P 4%	95.2%	55.2%	94.1%
VI 3% / P 9%	96.5%	68.6%	95.2%	VI 3% / P 9%	96.5%	67.7%	95.1%
VI 4% / P 16%	98.1%	84.0%	96.0%	VI 4% / P 16%	98.2%	85.3%	96.1%
VI 5% / P 25%	99.0%	93.3%	96.4%	VI 5% / P 25%	99.0%	93.3%	96.4%

5.3.3 Optimal Model - ReLU and Adam/SGD

Table 5.8 illustrates the Optimal Model's performance using the ReLU activation function and the Adam or SGD optimizers. As opposed to the results shown in Table 5.6 and Table 5.7, the model using ReLU activation performs differently depending on the optimizer. When using the Adam optimizer, performance is consistent. Even with minor increases, the model maintains a far better balance between recall and precision than the other configurations. Recall stays in the mid to high 90% and precision remains in the mid 80 to low 90%, with only a few exceptions. That said, with this structure, the Optimal Model produces poor results at the 9% increase and above mark (precision drops to the low 80%).

Table 5.8. Results for Optimal Model using ReLU Activation Function and either Adam or SGD Optimization on NAVFAC Data Sets.

ReLU								
Adam					SGD			
% Increase	Accuracy	Recall	Precision		% Increase	Accuracy	Recall	Precision
10%	97.7%	98.4%	81.7%		10%	98.9%	1.0%	90.4%
9%	96.9%	94.1%	78.9%		9%	97.7%	94.1%	84.3%
8%	98.8%	95.4%	92.5%		8%	98.0%	94.6%	86.6%
7%	98.7%	1.0%	88.9%		7%	99.6%	99.7%	96.4%
6%	97.5%	96.1%	81.8%		6%	98.2%	1.0%	85.1%
5%	99.0%	1.0%	91.1%		5%	99.6%	99.5%	96.6%
4%	98.2%	93.7%	88.7%		4%	98.4%	94.6%	90.3%
3%	98.5%	93.1%	89.5%		3%	97.4%	94.8%	81.6%
2%	98.5%	92.4%	93.0%		2%	97.3%	83.8%	88.8%
1%	97.2%	85.0%	86.6%		1%	94.6%	48.3%	93.3%
VI 2% / P 4%	98.8%	98.2%	90.8%		VI 2% / P 4%	97.3%	1.0%	78.6%
VI 3% / P 9%	97.4%	94.6%	81.9%		VI 3% / P 9%	96.6%	69.2%	95.2%
VI 4% / P 16%	97.5%	92.6%	84.0%		VI 4% / P 16%	98.2%	85.5%	96.1%
VI 5% / P 25%	98.9%	99.7%	90.6%		VI 5% / P 25%	99.0%	99.7%	91.1%

With the SGD optimizer, the model is worse at detecting minor increases and is inconsistent. At the 1% increase mark, the model outputs 48.3% recall score, compared to an 85% recall score with the Adam optimizer. However, the Optimal Model using ReLU and SGD produces acceptable results at the 5% increase point and above. Accuracy and recall remain in the high 90% while precision scores in the mid 80 to low 90%.

5.3.4 Deep Model - ReLU and Adam/SGD

Table 5.9 displays the Deep Model performance using the ReLU activation function and the Adam or SGD optimizers, respectively. As with the results shown in Table 5.8, the performance of the Deep Model depends on the optimizer. With the Adam optimizer, the model produces consistent results. Across all the data sets, accuracy remains in the high

90%, recall ranges from high 80 to mid 90%, and precision fluctuations from the low 80% to low 90%. This is the only configuration that does not produce any scores below 80% for accuracy, recall, or precision.

When the Deep Model uses the SGD optimizer and the ReLU activation function, it produces inconsistent results. The model fails to detect minor increases and produces worse than expected results when the FDIA percent increase rises above 5% (precision ranges from the mid to high 80s). The Deep Model with ReLU activation and SGD optimizer is the worst performer. It is the only configuration to fail to score a 90% in each metric for any data set. This configuration is the only one not suitable for use in our proposed IDS.

Table 5.9. Results for Deep Model using ReLU Activation Function and either Adam or SGD Optimization on NAVFAC Data Sets.

ReLU							
Adam				SGD			
% Increase	Accuracy	Recall	Precision	% Increase	Accuracy	Recall	Precision
10%	98.2%	96.1%	81.7%	10%	98.2%	97.8%	86.5%
9%	98.8%	98.7%	90.0%	9%	98.7%	98.0%	89.7%
8%	98.7%	93.7%	93.3%	8%	98.5%	96.5%	89.6%
7%	98.6%	93.1%	93.1%	7%	98.2%	96.5%	87.1%
6%	98.2%	94.8%	87.8%	6%	97.6%	96.5%	82.0%
5%	98.5%	89.8%	95.2%	5%	97.9%	96.7%	84.2%
4%	98.2%	96.1%	87.6%	4%	97.2%	95.6%	80.0%
3%	98.3%	93.5%	90.2%	3%	96.8%	95.9%	77.2%
2%	98.3%	93.7%	90.0%	2%	97.7%	92.0%	85.7%
1%	98.1%	94.8%	87.5%	1%	97.5%	84.5%	89.5%
VI 2% / P 4%	97.9%	94.6%	86.1%	VI 2% / P 4%	98.3%	95.4%	88.8%
VI 3% / P 9%	98.5%	96.1%	89.9%	VI 3% / P 9%	98.3%	95.4%	88.6%
VI 4% / P 16%	98.0%	95.9%	85.7%	VI 4% / P 16%	98.4%	97.4%	88.1%
VI 5% / P 25%	97.9%	99.5%	82.6%	VI 5% / P 25%	98.4%	98.4%	87.5%

5.4 Results Summary and Recommendation

The tables shown in the previous section show the potential benefit of using an autoencoder to detect malicious or anomalous activity in a smart grid environment. These results are consistent with other research in this area, and in many cases exceeds the previous research's results [4], [5], [11]. The key findings from this research are:

- Demonstrated that autoencoders can recognize full-scale attacks (targeting every input parameter) and select, more nuanced attacks (targeting specific input nodes).
- Showed the autoencoders' performance using real-world NAVFAC data.
- Presented two different models – the Optimal Model and the Deep Model and compared performance results.
- Demonstrated that the models' performance may vary depending on which activation function and optimizer are used.

The best autoencoder depends on stakeholder needs. Suppose the smart grid operator wants to detect minor fluctuations without receiving an unacceptably high number of false positives. In that case, the best autoencoder combines ReLU activation and Adam optimization. However, if the operator wants the model to detect significant attacks on the grid with the highest accuracy, recall, and precision, then the autoencoder should use sigmoid activation and Adam or SGD optimization. Another consideration is speed. As stated above, the Deep Model converges two to five times slower than the Optimal Model. Despite the deeper structure of the Deep Model, its performance was very similar to the Optimal Model. Therefore, a more complicated (deeper) model did not facilitate better results on this data set. The Deep Model has no significant benefit over the Optimal Model, making an autoencoder with the Optimal Model structure the best choice for the NAVFAC smart grid.

CHAPTER 6:
Conclusion

This thesis is intended to provide a foundation for additional research into unsupervised machine learning for the purpose of intrusion detection in the U.S. Navy smart grid. Understanding the threat of a power grid connected to the Internet, as well as the difficult nature of accurately classifying anomalous activity within a continual stream of normal data, highlights the importance of using machine learning.

6.1 Summary

To meet the energy modernization goals of the Secretary of the Navy, the Navy has expanded its attack surface by deploying smart grid technology across the fleet. To minimize the risks posed by cyber actors while maximizing power efficiency, an IDS using unsupervised machine learning must be utilized to protect critical infrastructure and sensitive networks. The real-world data sets used in this thesis were provided by NAVFAC, allowing us to experiment on actual AMI smart grid data. This thesis has demonstrated the effectiveness of autoencoder neural networks using TensorFlow for intrusion detection on the Naval smart grid.

This thesis begins by developing five different autoencoder models and testing their performance on an open-source Modbus data set. Accuracy, recall, and precision scores were our metrics of performance. The objective of this initial step was to build the best autoencoder to compare it against the autoencoder developed in [11], (referred to as the Deep Model in this thesis), on the NAVFAC smart grid data set. The model that performed the best on the Modbus data set was a two-layer network that compressed the first hidden layer by half and the second hidden layer to four nodes. This autoencoder is called the Optimal Model throughout this research.

To demonstrate the ability of anomaly detection using autoencoders, we tested an FDIA detection mechanism using the Optimal Model and Deep Model. To simulate an FDIA, we increased 10% of the NAVFAC data set from 1% to 10%. Experiments showed that the autoencoders could detect malicious activity against an FDIA on all AMI parameters and

when targeting a select few. Six model configurations are great candidates to include in the NAVFAC smart grid IDS architecture. The most appropriate model depends on the operator needs and threat environment. Furthermore, an essential contribution of this research is that the autoencoder learned the internal dependency of normal operation data, avoiding the need for labeled malicious data. This autoencoder feature helps overcome the unbalanced training data set problem in smart grid data sets.

6.2 Future Work

There are various avenues for future work in this area. First, it would be of benefit to test the autoencoder model on NAVFAC smart grid network traffic data set. The current NAVFAC data set contains AMI readings across a number of devices, classifying power fluctuations as anomalous or normal. However, there are several attack vectors against a smart grid that may not disturb the power supply. To this end, it would be beneficial to test whether an autoencoder can detect malicious activity in the NAVFAC smart grid network traffic.

We also think it would be prudent to test alternative unsupervised machine learning algorithms. While our autoencoder highlights the power of unsupervised machine learning, there are other machine learning algorithms that may be successful. By using the same data set, researchers could compare the performance of the competing algorithms, assisting end users in selecting the best model to match their needs.

Lastly, we believe an actual IDS implementation for the smart grid that incorporates unsupervised machine learning should be built. Extensive research has been conducted to develop anomaly detection algorithms. Less research has been produced outlining a prototype or proposal for an IDS implementation using machine learning. The U.S. Navy would benefit greatly from a working IDS.

APPENDIX A:
NAVFAC DATA SET INPUT FEATURES

1	Phase A Current
2	Phase B Current
3	Phase C Current
4	Neutral Current
5	Meter Frequency
6	Average kVAR
7	Average kiloWatts
8	Instantaneous kW
9	Maximum kVAR
10	Maximum kW
11	Phase A-B Voltage Phase Angle
12	Phase A Current Phase Angle
13	Phase A Voltage Phase Angle
14	Phase B-C Voltage Phase Angle
15	Phase B Current Phase Angle
16	Phase B Voltage Phase Angle
17	Phase C-A Voltage Phase Angle
18	Phase C Current Phase Angle
19	Phase C Voltage Phase Angle
20	Phase A Current THD
21	Phase A Voltage
22	Phase A-B Voltage
23	Phase B Voltage
24	Phase B-C Voltage
25	Phase C Voltage
26	Phase C-A Voltage

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
DATA PREPROCESSING SCRIPT

The following script imports the requisite libraries and models to perform machine learning. Then the data is manipulated to use within the TensorFlow framework.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import tensorflow as tf
import math
import sklearn as sk

from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import ModelCheckpoint,
                                TensorBoard, EarlyStopping
from tensorflow.keras import regularizers
from tensorflow.keras import optimizers
from tensorflow.keras import models

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.metrics import recall_score, classification_report, auc,
                                roc_curve, accuracy_score, precision_score
from sklearn.metrics import precision_recall_fscore_support, f1_score
from pandas.api.types import is_numeric_dtype

#Load Data
df = pd.read_excel('AMI_data.xlsx')
```

```
# Substitute NaN values with the mean of its column
column_means = df.mean()
df = df.fillna(column_means)

# Removes any values that are not numberical
numeric_cols = df.select_dtypes(exclude='number')
df.drop(numeric_cols, axis=1, inplace=True)

# Filter features from target
features = [f for f in list(df) if f not in ['Malicious']]

# Set training and test data
X_train, X_test, Y_train, Y_test = train_test_split(df[features],
                                                    df['Malicious'],
                                                    test_size=.5,
                                                    random_state=1)

# Parameter helps prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', mode='min',
                                        verbose=1, patience=50)

# Create set of normal traffic for training and testing

X_train_0 = X_train.copy()
X_train_0['Malicious'] = Y_train
X_train_0 = X_train_0[X_train_0['Malicious']==0]
X_train_0 = X_train_0.drop('Malicious', axis=1)

X_test_0 = X_test.copy()
X_test_0['Malicious'] = Y_test
X_test_0 = X_test_0[X_test_0['Malicious']==0]
X_test_0 = X_test_0.drop('Malicious', axis=1)
```

APPENDIX C:
BASIC MODEL SCRIPT

The following script defines the Basic Model autoencoder and sets its parameters.

```
# Auto encoder parameters
nb_epoch = 2000
batch_size = 128
input_dim = X_train_0.shape[1]
encoding_dim = math.ceil(input_dim / 2)
hidden_dim =      4
lr_e = 1e-2 # or 1e-3 when using SGD

# Create autoencoder model and set parameters
input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
    activity_regularizer=regularizers.l1(lr_e))(input_layer)
encoder = Dense(hidden_dim, activation="relu")(encoder)


decoder = Dense(hidden_dim, activation="relu")(encoder)
decoder = Dense(encoding_dim, activation="relu")(decoder)
decoder = Dense(input_dim, activation="linear")(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D:
DEEP MODEL SCRIPT

The following script defines the Deep Model autoencoder and sets its parameters.

```
# Auto encoder parameters
nb_epoch = 2000
batch_size = 128    # This was changed to match the article
input_dim = X_train_0.shape[1]
lr_e = 1e-2    # or 1e-3 when using SGD

# Create deep autoencoder model and set parameters
input_layer = Input(shape=(input_dim, ))
encoder = Dense(256, activation="relu",
    activity_regularizer=regularizers.l1(lr_e))(input_layer)
encoder = Dense(128, activation="relu")(encoder)
encoder = Dense(64, activation="relu")(encoder)
encoder = Dense(32, activation="relu")(encoder)

decoder = Dense(32, activation="relu")(encoder)
decoder = Dense(64, activation="relu")(decoder)
decoder = Dense(128, activation="relu")(decoder)
decoder = Dense(256, activation="relu")(decoder)
decoder = Dense(input_dim, activation="linear")(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E:
TRAINING AND TESTING SCRIPT

The following script compiles and runs the model. After that it establishes the threshold for normal data. Once the threshold is calculated, the model makes predictions on the test dataset.

```
# Compile and Run model
sgd = tf.keras.optimizers.SGD(learning_rate=0.001, decay=1e-6,
                                                momentum=0.9, nesterov=True)

# Compile model
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer=sgd)

# Save checkpoint to upload the best model for testing
cp = ModelCheckpoint(filepath="autoencoder_classifier_AMI.h5",
                     save_best_only=True,verbose=0)

history = autoencoder.fit(X_train_0, X_train_0,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(X_test_0, X_test_0),
                    verbose=1,
                    callbacks=[cp, tb, early_stop]).history

# Load the best performing parameters from training
autoencoder = tf.keras.models.load_model('autoencoder_AMI.h5')
```

```
# Establish threshold for normal traffic
x_train_pred = autoencoder.predict(X_train_0)
train_mae_loss = np.mean(np.abs(x_train_pred - X_train_0), axis=1)

# Calculate threshold by accounting for standard deviation
mean = np.mean(train_mae_loss, axis=0)
sd = np.std(train_mae_loss, axis=0)
final_list = [x for x in train_mae_loss if (x > mean - 2 * sd)]
final_list = [x for x in final_list if (x < mean + 2 * sd)]

#Set threshold to the max valvue within two standard deviations
sd_threshold = np.max(final_list)

# Make predictions for X_test and calculate the difference
test_x_predictions = autoencoder.predict(X_test)
test_mae_loss = np.mean(np.abs(test_x_predictions - X_test), axis=1)
```

APPENDIX F:
OUTPUT SCRIPT

The following script prints the accuracy score, displays a heat map confusion matrix, and generates a scatter plot illustrating the predictions in relation to the threshold and truth value.

```
# Display accuracy score
accuracy_score(Y_test, [1 if s > sd_threshold else 0
                                    for s in test_mae_loss])

# Graph depicts threshold and location of normal and malicious data
error_df_test = pd.DataFrame({'Reconstruction_error': test_mae_loss,
                              'True_class': Y_test})

error_df_test = error_df_test.reset_index()

groups = error_df_test.groupby('True_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error,
            marker='o', ms=3.5, linestyle='',
            label= "Malicious" if name == 1 else "Normal")
    ax.hlines(sd_threshold, ax.get_xlim()[0], ax.get_xlim()[1],
                            colors="r", zorder=100, label='Threshold')

ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
```

```
plt.show();

#Confusion Matrix heat map

pred_y = [1 if e > sd_threshold else 0 for e in
            error_df_test['Reconstruction_error'].values]
conf_matrix = confusion_matrix(error_df_test['True_class'], pred_y)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix,
            xticklabels=["Normal","Malicious"],
            yticklabels=["Normal","Malicious"],
            annot=True, fmt="d");
plt.title("Confusion_matrix")
plt.ylabel('True_class')
plt.xlabel('Predicted_class')
plt.show()

#Print accuracy, recall, and precision scores
print("accuracy:", accuracy_score(error_df_test['True_class'], pred_y))
print("recall:", recall_score(error_df_test['True_class'], pred_y))
print("precision:", precision_score(error_df_test['True_class'], pred_y))
```

List of References

[1] Department of the Navy, "Department of the Navy's energy program for security and independence," Oct. 2010 [Online]. Available: https://www.secnav.navy.mil/eie/ASN%20EIE%20Policy/Naval_Energy_Strategic_Roadmap.pdf

[2] Energy Central, "NAVFAC reaches key milestone in deploying new smart energy monitoring and control solution," Apr. 2019 [Online]. Available: https://www.navfac.navy.mil/news/2019_press_releases/April2019-press-releases/NAVFAC-Reaches-Key-Milestone-in-Deploying-New-Smart-Energy-Monitoring-and-Control-Solution.html

[3] Department of Energy, "The Smart Grid." Available: https://www.smartgrid.gov/the_smart_grid/smart_grid.html

[4] C. A. Schiesser, "Malicious threat detection for the NAVFAC-based smart grid network using bayesian classification and machine learning," M.S. thesis, Dept. of Computer and Electrical Engineering, NPS, Monterey, CA, USA, 2020 [Online]. Available: https://calhoun.nps.edu/handle/10945/66136

[5] V. C. Chan, "Using a k-nearest neighbors machine learning approach to detect cyberattacks on the Navy smart grid," M.S. thesis, Dept. of Computer and Electrical Engineering, NPS, Monterey, CA, USA, 2020 [Online]. Available: https://calhoun.nps.edu/handle/10945/66054

[6] K. Kimani, V. Oduol, and K. Langat, "Cyber security challenges for IoT-based smart grid networks," *International Journal of Critical Infrastructure Protection*, vol. 25, pp. 36–49, 2019.

[7] M. Ouaissa and M. Ouaissa, "Cyber security issues for IoT based smart grid infrastructure," in *IOP Conference Series: Materials Science and Engineering*, no. 1. IOP Publishing, 2020, vol. 937, p. 012001.

[8] Z. El Mrabet, N. Kaabouch, H. El Ghazi, and H. El Ghazi, "Cyber-security in smart grid: Survey and challenges," *Computers & Electrical Engineering*, vol. 67, pp. 469–482, 2018.

[9] T. Baumeister, "Literature review on smart grid cyber security," *Collaborative Software Development Laboratory at the University of Hawaii*, vol. 650, 2010.

[10] D. B. Rawat and C. Bajracharya, "Detection of false data injection attacks in smart grid communication systems," *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1652–1656, 2015.

[11] C. Wang, S. Tindemans, K. Pan, and P. Palensky, "Detection of false data injection attacks using the autoencoder approach," in *2020 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. IEEE, 2020, pp. 1–6.

[12] M. A. Mustafa, N. Zhang, G. Kalogridis, and Z. Fan, "DEP2SA: A decentralized efficient privacy-preserving and selective aggregation scheme in advanced metering infrastructure," *IEEE Access*, vol. 3, pp. 2828–2846, 2015.

[13] A. Lemay and J. M. Fernandez, "Providing SCADA network data sets for intrusion detection research," in *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*, 2016.

[14] J. Jordan, "Introduction to autoencoders," Data Science, Mar. 19, 2018 [Online]. Available: https://www.jeremyjordan.me/autoencoders/

[15] Sayantini, "Autoencoders tutorial: What are autoencoders?" Edureka, May 14, 2020 [Online]. Available: https://www.edureka.co/blog/autoencoders-tutorial/

[16] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, 2014, pp. 4–11.

[17] S. Zavrak and M. Iskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, pp. 108 346–108 358, 2020.

[18] P. Kim, *Matlab deep learning with machine learning, neural networks and artificial intelligence*. New York, NY, USA: Apress, 2017.

[19] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese control and decision conference (CCDC)*. IEEE, 2018, pp. 1836–1841.

[20] W. A. Vega, "Neural networks for constrained optimization," M.S. thesis, Dept. of Operations Research, NPS, Monterey, CA, USA, 2020 [Online]. Available: https://calhoun.nps.edu/handle/10945/66040

[21] N. S. Keskar and R. Socher, "Improving generalization performance by switching from Adam to SGD," *arXiv preprint arXiv:1712.07628*, 2017.

[22] L. Wan, Z. Zhang, and J. Wang, "Demonstrability of narrowband internet of things technology in advanced metering infrastructure," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–12, 2019.

[23] M. Z. Huq and S. Islam, "Home Area Network technology assessment for demand response in smart grid environment," *20th Australasian Universities Power Engineering Conference*, vol. 2010, pp. 1–6, 2010.

[24] TensforFlow, "TensorFlow," 2022 [Online]. Available: https://www.tensorflow.org/

[25] M. Gharib, B. Mohammadi, S. H. Dastgerdi, and M. Sabokrou, "Autoids: auto-encoder based method for intrusion detection system," *arXiv preprint arXiv:1911.03306*, 2019.

[26] P. Schneider and K. Böttinger, "High-performance unsupervised anomaly detection for cyber-physical system networks," in *Proceedings of the 2018 workshop on cyber-physical systems security and privacy*, 2018, pp. 1–12.

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