4-2018

Enhancing Firewall Filtering Performance Using Neural Networks

Heba Saleous

UAEU

جامعة الإمارات العربيـة المتحدة
United Arab Emirates University

United Arab Emirates University

College of Information Technology

Department of Information Systems and Security

ENHANCING FIREWALL FILTERING PERFORMANCE USING NEURAL NETWORKS

Heba Saleous

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Security

Under the Supervision of Dr. Zouheir Trabelsi

April 2018

Declaration of Original Work

I, Heba Saleous, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled "*Enhancing Firewall Filtering Performance Using Neural Networks*", hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Zouheir Trabelsi, in the College of Information Technology at UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature: [illegible] Date: May 20, 2018

Advisory Committee

1) Advisor: Dr. Zouheir Trabelsi

Title: Associate Professor

Department of Information Systems and Security

College of Information Technology

2) Member: Dr. Mohammad Mehedy Masud

Title: Associate Professor

Department of Information Systems and Security

College of Information Technology

3) Member: Dr. Huweida Said

Title: Associate Professor

Zayed University, Dubai, U.A.E.

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Dr. Zouheir Trabelsi
   Title: Associate Professor
   Department of Information Systems and Security
   College of Information Technology

   Signature [illegible] Date May 9th, 2018

2) Member: Dr. Mohammad Mehedy Masud
   Title: Associate Professor
   Department of Information Systems and Security
   College of Information Technology

   Signature [illegible] Date 09/05/2018

3) Member (External Examiner): Dr. Huwaida Said
   Title: Associate Professor
   Department of Technological Innovation
   Institution: Zayed University, Dubai, U.A.E.

   Signature [illegible] Date May 8, 2018

This Master Thesis is accepted by:

Dean of the College of Information Technology: Dr. Khaled Shuaib

Signature [illegible] Date 20/5/2018

For Dean of the College of Graduate Studies: Professor Nagi T. Wakim

Signature [illegible] Date 20/5/2018

Copy 7 of 7

Abstract

The internet has grown to a point where people all over the world have become dependent on this convenient communication medium. However, with this dependency, malicious traffic has become a major concern. Consequently, firewalls have become a mandatory part of any network, due to their ability to filter the traffic based on rules that state which packets should be accepted or denied. However, filter rules must be manually configured by a network administrator, and packets that do not fit any rule may be subject to wrong judgement by the firewall. Neural networks can learn the filter rules in order to decide if packets that do not fit any specific rules should be accepted or denied. The neural network will be trained with existing packet data and their firewall actions, and then tested to determine the amount of correctly classified packets compared to the firewall.

Keywords: Network Security, Firewall, Packet filtering, Neural Networks

Title and Abstract (in Arabic)

تعزيز أداء جدار الحماية باستخدام الشبكات العصبية

الملخص

لقد نمى استخدام الإنترنت إلى حد أصبح فيه الناس في جميع أنحاء العالم يعتمدون على وسيط الاتصال المريح هذا الذي تم توفيره. و مع هذه التبعية ، أصبحت حركة الاتصالات الضارة مصدر قلق كبير. ولهذا السبب، تكون جدران الحماية جزءًا إلزاميًا من أي شبكة رقمية، نظرًا لقدرتها على تصفية الاتصالات استنادًا إلى القواعد التي تنص على قبول الرزم أو رفضها. ويتطلب هذا تهيئة قواعد التصفية يدويًا بواسطة مسؤول الشبكة ، كما وقد تخضع الرزم التي لا تلائم أي قاعدة لحكم خاطئ بواسطة جدار الحماية. يمكن للشبكات العصبية معرفة قواعد التصفية التي تم تعيينها من قبل المسؤولين من أجل تحديد ما إذا كانت الرزم التي لا تناسب أي قواعد محددة يجب قبولها أو رفضها. سيتم تدريب الشبكة العصبية مع بيانات الرزمة الحالية و قواعد جدار الحماية ، ثم يتم اختبارها لتحديد دقة التصفية مقارنة بجدار الحماية.

مفاهيم البحث الرئيسية: امن الشبكة، جدار الحماية، تصفية الرزم، الشبكات العصبية

Acknowledgements

I would like to thank my advisor, Dr. Zouheir Trabelsi, for his constant support and motivating attitude during my time as his student. I am grateful for his patience and expert knowledge. Dr. Zouheir always encouraged me to work hard and at my best, helping me find solutions and accomplish tasks.

I would also like to thank Dr. Mohammed Masud, whose knowledge of Machine Learning helped me with any misunderstandings I had with the subject. His advice helped me clearly understand the real issue being tackled by this thesis.

Dedication

To my father, Dr. Nazmi Saleous, whose love and support has fueled my quest for knowledge, and whose character has inspired me to become the best I can be.
A true role model.

Table of Contents

Title i
Declaration of Original Work ii
Copyright iii
Advisory Committee iv
Approval of the Master Thesis v
Abstract vii
Title and Abstract (in Arabic) viii
Acknowledgements ix
Dedication x
Table of Contents xi
List of Tables xiii
List of Figures xiv
List of Abbreviations xv
Chapter 1: Introduction 1
Chapter 2: Research Problem 3
    2.1 Objectives 3
Chapter 3: Background 5
    3.1 Literature Review 5
    3.2 Neural Networks 7
Chapter 4: Methodology 17
    4.1 Materials 17
        4.1.1 Types of Neural Networks 17
        4.1.2 Programming Language Selection 18
    4.2 Design 19
Chapter 5: Testing and Results 24
    5.1 With Default DENY Rule 26
        5.1.1 Four (4) Rules 26
        5.1.2 Nine (9) Rules 29
        5.1.3 Fifteen (15) Rules 32
    5.2 With Default ACCEPT 34
        5.2.1 Four (4) Rules 34
        5.2.2 Nine (9) Rules 36
        5.2.3 Fifteen (15) Rules 38
    5.3 Uncertainties 40

5.3.1 Four (4) Rules 41
5.3.2 Nine (9) Rules 42
5.3.3 Fifteen (15) Rules 42
5.4 Observation 43
Chapter 6: Discussion 45
6.1 Advantages 45
6.2 Disadvantages 45
6.3 Recommendations 47
Chapter 7: Conclusion 49
References 51

List of Tables

Table 1: The amount of training and testing examples 24

List of Figures

Figure 1: A graph of the input and the output 8
Figure 2: Gradient descent to find the value of θ the converges 9
Figure 3: The effects on the gradient descent 10
Figure 4: Three models that have been fitted to a training set 12
Figure 5: The basic structure of an ANN 13
Figure 6: Flowchart of the packet generation phase 20
Figure 7: Flowchart of the simulated firewall 21
Figure 8: Flowcharts of the ANN for the (a) training and (b) testing phases 22
Figure 9: The initialization of the ANN parameters 23
Figure 10: The randomly generated firewall rules 26
Figure 11: The training time of the ANN (4 rules) 27
Figure 12: The ANN's error throughout the training phase 27
Figure 13: Comparison of the percentage of correctly classified samples 28
Figure 14: Network information with 9 rules 29
Figure 15: The amount of time spent training the ANN 30
Figure 16: The ANN's error throughout the training phase 30
Figure 17: Comparison of the percentage of correctly classified samples 31
Figure 18: Network information with 15 rules 32
Figure 19: The training time of the ANN with different amounts of examples 32
Figure 20: The training errors generated 33
Figure 21: Comparison of the percentage of correctly classified samples 34
Figure 22: The training times of the ANN 35
Figure 23: The errors calculated during the training phase 35
Figure 24: Comparison of the percentage of correctly classified samples 36
Figure 25: The training times with varying amounts of training examples 37
Figure 26: The calculated training errors during the ANN's training phase 37
Figure 27: Comparison of the percentage of correctly classified samples 38
Figure 28: The training time of the ANN 39
Figure 29: The training errors generated during the ANN's training phase 39
Figure 30: Comparison of the percentage of correctly classified samples 40
Figure 31: The differences in successful classifications (4 rules) 41
Figure 32: The differences in successful classifications (9 rules) 42
Figure 33: The differences in successful classifications (15 rules) 43

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FANN	Fast Artificial Neural Network
FFNN	Feed Forward Neural Network
FW	Firewall
IDS	Intrusion Detection System
LAN	Local Area Network
ML	Machine Learning
RNN	Recurrent Neural Network
SVM	Support Vector Machine

Chapter 1: Introduction

In the current era, the primary medium of communication has become the internet. It allows people to communicate with each other all over the world in a matter of seconds. Network connectivity is required to achieve this, making it a necessity in everyone’s lives. Due to its convenient and accessible nature, the internet is open not only to professionals, but also to children and computer neophytes in general. However, the internet is also open to those with malicious intentions, providing an entirely new and convenient medium for attacks.

Just as network connectivity has become a necessity in the lives of the human population, so has network security. Without network security, those with ill intentions would be free to commit cybercrimes, whether they are against other people or organizations. Attacks can range from minor incidents, such as spam mail, to extreme attacks that may results in physical harm to others. An example of the latter is the attack on the German steel mill that occurred in 2014. The attackers were able to gain access to the plant’s network by gaining information through a spear-phishing email [1]. This information caused critical process components from functioning properly, resulting in physical damage to the mills.

One major tool used to secure a network is the firewall. A network firewall is a system that enforces access control policies, usually in the form of rules, to control the traffic that enters a network or machine [2]. The packets being transported are filtered based on certain characteristics, usually their source IP addresses, source and destination ports, and the protocol being used. The rules set in a firewall are configured manually by the network’s administrator and are set based on known information about the incoming and outgoing traffic. Packets that are entering a network from known malicious sources are blocked and prevented from entering the network. Alternatively, connections that are mandatory for daily functions in an organization, such as those related to checking emails, can have rules allowing them.

While network security is advancing and becoming a necessity all over the world, Artificial Intelligence (AI) and Machine Learning (ML) have also become major fields of research. The applications of these topics extend to any professional field, from art and media to medicine. Machine Learning is a field of study within Artificial Intelligence that allows computers to learn from existing examples without being manually programmed [3], [4]. The examples used for training includes the desired outputs, as well as the inputs that led to them.

One common technique of Machine Learning is the Neural Network. These are networks that draw inspiration from the human brain’s synapses, linking inputs together to produce an output based on known and learned information. Artificial neural networks (ANNs) have been used in multiple applications, such as ailment diagnosis and speech recognition. With the rising popularity of this field of study and the amount of research going into it, people are beginning to find more uses for ANNs, even if it is just for entertainment. According to [2], the application of artificial intelligence in the area of intrusion detection already exists. For example, exploring techniques that can be used to design, implement, and enhance existing intrusion detection systems.

Chapter 2: Research Problem

Firewalls have become an important, mandatory part of any network, whether they are set up in a home or organization network. However, the presence of the default rule might cause some packets to be filtered incorrectly. Harmless packets blocked by the default DENY rule may cause some inconvenience for users when trying to access popular services. Malicious packets that are passed into a network by a default ACCEPT rule will put the network and its users at risk of attack.

ANNs may find a home in these networks, providing a more trustworthy filtering mechanism than the firewall's default rule. Given the possible uses of ANNs, their use in network security may improve the firewall's filtering capabilities. The ANN's purpose, in this case, is to learn the firewall's filter rules, as well as the nature of the incoming traffic. Based on this input, the ANN will re-filter traffic that would normally be subjected to the default rule in a firewall.

2.1 Objectives

Before the ANN can be designed to address the research problem, clear goals need to be considered in order to understand how a contribution to any existing work can be made. The main objective of this thesis is to design and develop an ANN that can work alongside a firewall in order to improve its packet filtering capabilities. This can be done by doing the following:

I. Understanding the functionality of an ANN and how it may improve firewall packet filtering performance

II. Design and implement an ANN that works alongside a firewall, learning about the firewall's behavior and the nature of its usual network traffic, and efficiently reacting to that traffic

Once these goals have been met, experiments with specific conditions will be run to observe the behavior of the ANN. The results of the experiments to be performed will determine how well an ANN can work with packet data. Any modifications the system may require after observing these results will also be determined in order to improve the system.

Chapter 3: Background

3.1 Literature Review

Valentin and Maly [2] investigated the use of ANNs with firewalls with two scenarios. The first involved building a firewall by using a sample of the normal traffic that would pass through a network, and the second scenario focused on investigating the possibility of copying the rules and behaviors of a currently existing firewall [2]. The rules were encoded into the neural network so that it could follow the same functionality of the standard firewall used; packets would be allowed or denied depending on the given filter rules. The data sets generated included packets that had a corresponding rule in the given filter and followed an 80:20 ratio for denied and accepted packets. However, the packets that were generated were done such that they were controlled, rather than random.

ANNs have been found to be useful in the implementation of Intrusion Detection Systems (IDSs). IDS designers tend to use ANNs as a pattern recognition technique, matching outputs to certain inputs. If the input data presented a system does not match the learned model, the ANN will provide an output based on what the network was taught. The author, Reddy, of [5] described techniques, one based on supervised learning and the other on unsupervised learning, for the neural network: Multilayered Feed-Forward Neural Networks and Kohonen’s Self-Organizing Maps, respectively. The author also discusses an ANN’s ability to learn a system over time, rather than basing activity on a signature of normal behavior. This is useful for anomaly recognition in misuse detection systems. ANNs are versatile, inherently fast, can predict when an attack may happen, and can learn to identify new misuse patterns. Although the benefits of using ANNs in misuse detection have been mentioned, no actual implementation was provided by the author.

Ussath, et al. [6] employed ANNs to detect suspicious behavior over a network. In this case, the neural network must be able to differentiate between normal and abnormal user behavior. To do this, the authors determined key behavioral features that need to be logged. These features are the date, time, session duration, and the system used for the user login. The neural network, however, was designed to analyze behavior with simulated data because existing data is difficult to use. The difficulty arises from companies maintaining confidentiality for both log events and user actions due to privacy concerns. Yet, even with these limitations, the authors concluded that ANNs could, indeed, be used for behavior analysis, and the approach used can be applied to real-world use cases.

Wang [7] proposed a scheme to enhance packet classification using multiple decision trees. The author explored multiple decision tree algorithms to contrast their own algorithm. The algorithms explored were Hierarchical Intelligent Cuttings (HiCuts), its extension HyperCuts, a third algorithm that used partial filter bits rather than the specifications of fields like the preceding two algorithms, and EffiCuts. The authors aimed to create an algorithm that would not rely on hardware support like previous algorithms. Real and synthetic filter databases were used to test the algorithm. The author was able to conclude that the multiple decision tree algorithm scaled well in terms of speed and storage performance.

The authors of [8] explored of Support Vector Machines (SVMs) to create User Profile Filters in order to examine traffic signatures. Four predefined traffic metrics were monitored: Total bytes, total packets, destination socket, and destination port. The authors found that using these four metrics for the user profile filter not only allowed them to monitor user activity to detect attacks, but to also detect anomalies in the network. Network traffic was sniffed and decoded to analyze the header and payload. Data packets were then isolated based on their source to examine the signature. The SVM was used to create a normal profile for users in order to determine if data is normal or an anomaly. The authors were able to deduce that using an SVM with a network traffic prediction technique and a flexible packet filter resulted in only 0.5% of the traffic being misclassified with a false alarm rate that did not exceed 3%.

3.2 Neural Networks

Before using an ANN into any project, it is important to understand its concepts and the calculations that occur in the background. There are two types of learning processes that an ANN can undergo in the application it is used for: Supervised and Unsupervised Learning. In supervised learning, the data sets to be used by the neural network are provided, and an idea of the desired output is available [9]. Unsupervised learning is the opposite; the desired output is unknown with no or little data given.

While designing an ANN, *features* need to be taken into consideration. These are the different types of input in a system that the neural network should expect. Each feature will be given a weight, and then used in a function known as the hypothesis function, shown below [9]:

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots \theta_n x_n \tag{1}$$

In this formula, *x* is the value given to a specific feature *n*, and θ is the weight that is given to that feature. The weights are selected such that the result is close enough to the desired output with the data that has already been given. The purpose of the hypothesis function is to map input values to certain outputs [9]. This hypothesis will be used during the calculation of the cost function, which is a measure of the error in the neural network's ability to estimate a relationship between the input values and the known output values [9], [10], [11]. The error cost function is given in Formula 2.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) - y^{(i)})^2 \tag{2}$$

In this equation, m is the number of training examples in the data set that is being used, x, as before, is the input feature, y is the known output, and i represents the number of the training example in the current iteration. In order for the neural network to be accurate in its future calculations, the result of the error cost function will need to be as small as possible. If the result of this function is considered to be too large, the weights θ should be adjusted in the hypothesis such that the resulting value of the hypothesis h should be as close as possible to the output value y [9], [10].

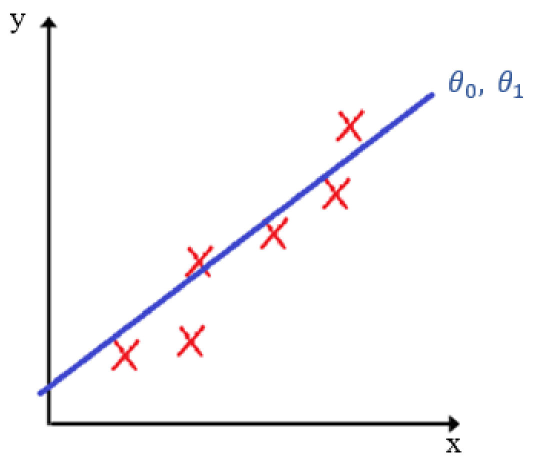


Figure 1: A graph of the input and the output [9]

The values of θ will need to be modified so that the graph of the outputs produces a line that fits as closely to the training examples as possible, as seen in Figure 1. This is typically known as Linear Regression, where independent inputs are given a linear relationship to the dependent outputs [9], [10]. In order to achieve this, the cost function will need to be minimized through a process known as Gradient Descent. This process can be generally defined as “an iterative optimization procedure” where each step improves the result “by taking a step along the negative of the gradient of the function to be minimized at the current point” [11]. The values of θ can be fixed to achieve this by using Formula 3.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) - y^{(i)}) x_j^{(i)} \tag{3}$$

In this formula, α is known as the learning rate, which is the rate at which the error is reduced [9], [10]. The subscript *j* represents the identification number of the variable, or feature, for which this calculation is occurring. When a large number of features are used in a model, the values of *x* and θ are vectorized so that:

$$h_\theta(x) = [\theta_0 \quad \theta_1 \cdots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x \tag{4}$$

Figure 2 shows how gradient descent appears with regards to a single weight θ and the cost function *J*(θ).

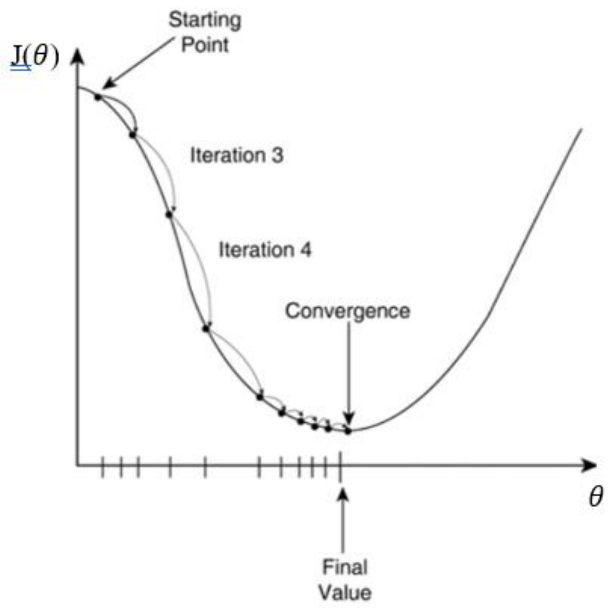
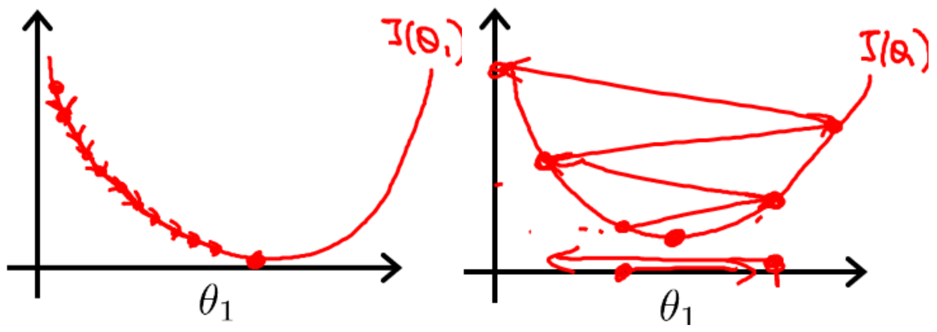


Figure 2: Gradient descent to find the value of θ the converges [10]

The end goal of Gradient Descent is to find a local minima, or a point of convergence. It is important to note that if the learning rate used in gradient descent is too large, the steps taken in each iteration may be too large and may miss the local minima [9], [10]. Alternatively, if the learning rate is too small, gradient descent may be very slow, which will affect the neural network later on by slowing down its training phase.



(a) (b)

Figure 3: The effects on the gradient descent when the learning rate is (a) too small and (b) too large [9]

Linear Regression works when the output value is expected to be continuous. However, ANNs have the ability to classify data, grouping the data based on the output that was calculated. When the classification problem is introduced, the Logistic Regression model is used when the possible output values are discrete values [9], [11]. For this model, the hypothesis is modified as follows:

$$h_\theta(x) = g(\theta^T x), \text{ where } g(\theta^T x) = \frac{1}{1+e^{-(\theta^T x)}} \tag{5}$$

In this formula, $g(\theta^T x)$ is known as the Sigmoid Function, or the Logistic Function. Using the Sigmoid Function, the hypothesis $h_\theta(x)$ will instead calculate the probability of the "class" of the output [9], [11]. In order to decide which "class" the output belongs to, a decision boundary is introduced by the hypothesis function, where certain hypothesis values will determine whether the output will belong to one class or another [9].

Modifying the hypothesis to be suitable for logistic regression will require a modification to the cost function used as well. The new cost function is given as [9], [12]:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} f\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) \tag{6}$$

$$\text{where } f\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\log\left(h_\theta^{(i)}(x)\right)$$

However, this function works when there are a very small number of possible classifications for the output of the model. In order to work in larger multi-class models, the cost function will need to be further modified. The modified cost function can be written as the following [9], [12]:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \log(1 - h_\theta\left(x^{(i)}\right))] \qquad (7)$$

In order to be compatible with the larger, vectorized data sets found with most ANNs this modified equation can be translated to [9]:

$$J(\theta) = \frac{1}{m} \cdot (-y^T \log h) - (1 - y) \log(1 - h)) \qquad (8)$$

The y^T in this case is the transpose of the vector containing the output values of the training examples.

Despite all of the effort that goes into optimizing the cost function to increase the efficiency of the neural network and its calculations, two other issues need to be addressed: underfitting and overfitting. Underfitting occurs when the hypothesis algorithm fails to map the trend of the training data accurately, likely due to using insufficient features in the model [9], or when “the model is not able to obtain a sufficiently low error value on the training set” [13]. The opposite, overfitting, occurs when the algorithm fits the trend of the data too well, causing many curves [9], [11], [14]. This may cause the neural network to “learn noise and spurious relations” [15] and result in a failure to generalize [9], [12], [15]. Failing to generalize means that the training examples have been “memorized” and the network will fail to accept new data. The issue of overfitting typically occurs when there are too many input features or complex functions that are causing curves that may not relate to the data being fed [9], [13], [15]. An example of how underfitting and overfitting appear with a training set can be seen in Figure 4.

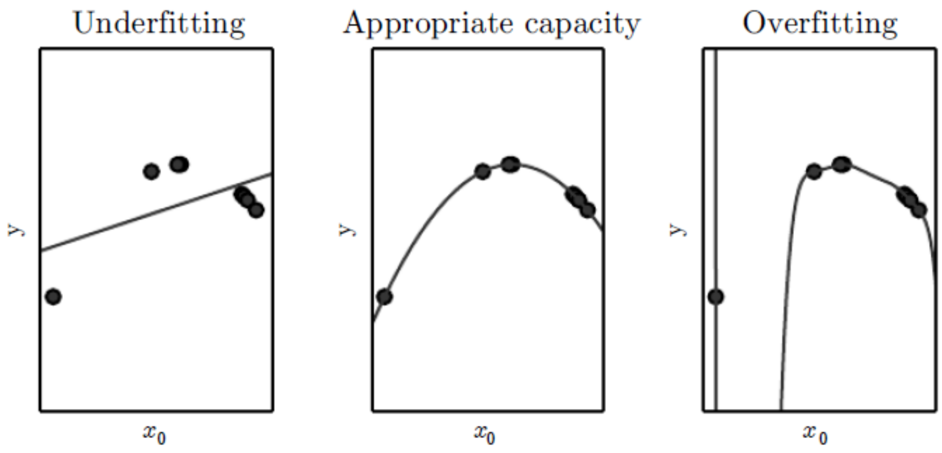


Figure 4: Three models that have been fitted to a training set [13]

Overfitting can be resolved either by reducing the number of features used for a model, or by using regularization functions. Regularization functions can act as stabilizers for learning algorithms [11]. The main purpose of a regularization function is to reduce the generalization error while leaving the training error unaffected [13]. To regularize the model, an extra term is added to the general cost function, modifying it as follows [9], [11], [13]:

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2] \tag{9}$$

In the formula above, λ is known as the regularization parameter. Its purpose is to determine the inflation of the weights θ and smooth out the model [9], [11]. The purpose of adding the second summation, $\sum_{j=1}^{n}\theta_j^2$, is to make the parameters of the equation smaller, since the value of the regularization parameter is expected to be higher. Setting the value of the regularization parameter too high, however, may end up causing the model to underfit [9], [13]. Should the cost function need to be adjusted, the gradient descent formula can also be modified for regularization as follows:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j \tag{10}$$

All of these calculations occur in machine learning. For ANNs, as previously mentioned, the values for the parameters and the weights are typically fitted into vectors. All of the calculations that are performed are used in the design and implementation of an ANN.

The design of an ANN takes inspiration from the human nervous system's nodes and synapses. The basic structure of an ANN can be seen in Figure 5 below.

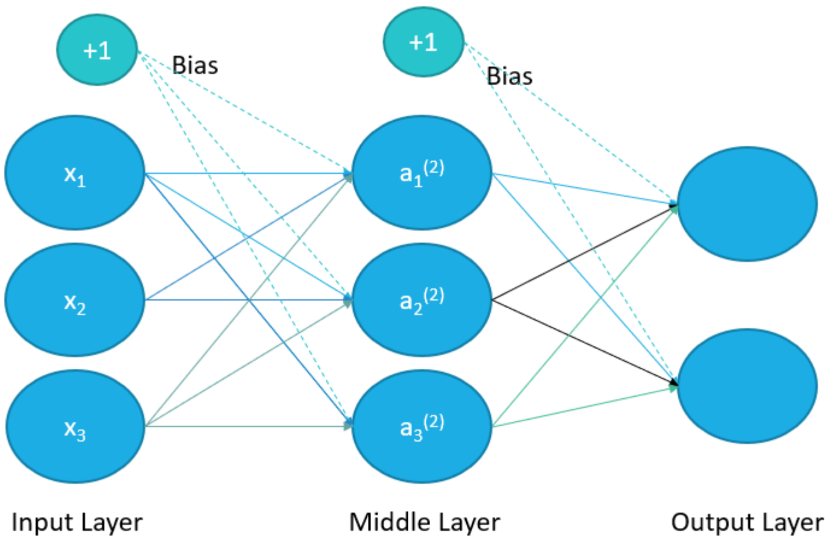


Figure 5: The basic structure of an ANN

Certain inputs will lead to different paths depending on values given, which will then be used to determine the output. They can be defined as mathematical methods for mapping inputs to certain outputs [16].

An ANN consists of at least three layers: the input layer, the middle, or hidden, layer(s), and the output layer. Each layer consists of neurons, which represent values with "activations" [17]. For the input layer, the activation of each neuron is simply the value of the input itself. However, the activation of the neurons in the middle layer(s) require the values of the previous layer, either the input layer or another middle layer, to pass through an activation function, which uses the Sigmoid function. The activation values can be calculated using the following formula [9]:

$$a_i^{(j)} = g\left(\Theta_{i\,0}^{(j)} x_0 + \Theta_{i\,1}^{(j)} x_1 + \dots + \Theta_{i\,n}^{(j)} x_n\right) \tag{11}$$

In this formula, *i* represents the number of the unit, *j* represents the number of the layer, and Θ represents the "matrix of weights controlling function mapping from layer *j* to layer *j*+1" [9]. During the activation function calculations, each value must be seen as a vector or a matrix [16]. The ANN can then be imagined as a connection of vectors. Figure 5 can be reimagined as the following:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{bmatrix} \rightarrow h_\theta(x)$$

It is important to note that the input and middle layers may contain bias nodes and values, which are denoted by x_0 and $\Theta_0^{(j)}$. The purpose of the bias node is to allow the shift of the activation function as needed by allowing the ANN to decide what gets multiplied with the constant terms [9]. The bias values typically take a value of 1 but can be assigned a different value depending on the design of the ANN.

In order to train an ANN, the formulas that have been discussed so far are used in a method known as BackPropagation, or BackProp for short. This method is used to train multi-layer ANNs [13], [18], [19] by calculating the cost function and allowing it to flow backwards through the ANN in order to then compute the gradient [9], [11], [13]. The computed gradient is then used to perform gradient descent for the actual learning process of the ANN [13]. The following steps occur in the BackProp process [9], [19], [20]:

1) Feed-Forward Computation
2) BackProp the output layer
3) BackProp the middle layer
4) Update the weights

In the first step, the ANN moves forward from the input layer to the output layer and calculates the values of all of the nodes on the way. When BackPropagating the output layer, the errors from the output nodes to the middle layer nodes are calculated. The error is calculated with the following [19], [20]:

$$Error = (node\ value) * \big(1 - (node\ value)\big) * ((desired\ value) - (node\ value) \qquad (12)$$

It should be noted that the *node value* in this case is the value of the output layer node. The rate of change is then computed with the following formula:

$$Rate\ of\ Change\ =\ (learning\ rate)\ *\ (error\ value)\ *\ (middle\ layer\ node\ value) \qquad (13)$$

The next step is to BackProp to the middle layer, where the error from the middle layer to the input layer, or any other preceding middle layer, is calculated. The formula used is as follows:

$$Middle\ Layer\ Error = (Rate\ of\ Change) * (Output\ Node\ Error) \qquad (14)$$

Once the middle layer error has been computed, the rate of change is also computed using the same formula as the previous step. If there are multiple middle layers, this process repeats until the error and rate of change of the first middle layer nodes are calculated. Once all of the error values have been evaluated, the weights throughout the ANN are calculated and updated in order to reduce the error. The new weights can be computed using the following formula:

$$New\ Weight = (old\ weight) + (rate\ of\ change) + \big((momentum) * (previous\ weight\ change)\big) \qquad (15)$$

The momentum is the value in an ANN that allows it to stabilize. The value of the *previous weight change* is always going to be 0 for the first time the ANN goes through BackProp. Once all of the errors and new weight values have been calculated, the weights throughout the ANN can be updated. Once the weights have been updated, the error rates can be recomputed to ensure that the new weight values have indeed reduced the error throughout the ANN. This will allow the ANN to learn more efficiently and provide more accurate results, increasing the likelihood that the correct output value will be predicted during the testing phase.

All of the calculations that occur for ANNs are not done by hand, as they can become quite complex and difficult to solve. The calculations are done through programming in the user’s language of choice. Alternatively, they can also be done in Octave and then integrated in the ANN’s code.

Chapter 4: Methodology

4.1 Materials

Before the proposed system can be designed, the materials and resources available and required need to be taken into consideration. Having an idea of what is available for use can result in having a smoother design phase once decisions about which resources to use have been made.

4.1.1 Types of Neural Networks

There are different types of ANNs that exist, each with their own purposes. The three types that were taken into consideration were Convolutional, Feed-Forward, and Recurrent ANNs. In order to decide which type should be used for this thesis, the uses and benefits of each type were considered.

Convolutional ANNs (CNN) are mostly used for processing images, videos, and audio [21], [22], [23]. The purpose of this network is to take the media as its input and returns output that has been classified [21]. These networks are mostly used in applications involving computer vision [23].

Feed-Forward ANNs (FFNN) are a basic kind of ANN that involve layers (input, middle, and output) that are directly connected to each other in an acrylic manner [13], [18], [21]. They are straightforward ANNs that move in a single direction, from the input layer to the output layer. There is no recurrence or feedback, which may occur in other types of ANNs [13], [21].

Recurrent ANNs (RNN) are a type of FFNN that processes sequential data [13] and includes a time factor [21]. The difference between FFNNs and RNNs is that the latter is able to share parameters across the model, while an FFNN is a straightforward model [13]. Parameter sharing allows the model to extend itself and consider pieces of information that can occur in multiple possible places [13]. This can be useful for applications that try to complete or predict information, such as autocompletion [21].

For this thesis, an ANN is needed to process data that relates directly to a single output. CNNs having fewer parameters, making them easier to train [22]. For this project, however, since none of the input is categorized as any of the media that a CNN typically takes, using a CNN is not an option. An RNN processes sequential data that are not only from one single run but may also include data from the previous run as well [21]. Doing this means that one order of data may yield different results than if the same data was fed in a different order. Given the nature of the data being fed to the ANN in this thesis, RNNs may not be the suitable choice. Although FFNNs are basic and simple, the data being fed to the network will be handled in a straightforward manner. The application to be discussed in this thesis does not require any feedback or recurrent behavior. Therefore, the ANN type of choice for this thesis is the Feed-Forward ANN.

4.1.2 Programming Language Selection

For the ANN, the main programming languages used for development are C/C++, Java, and Python. For the purpose of this thesis, the language used to program the ANN is C.

The reason this language was selected was to use the Fast Artificial Neural Network (FANN) library, which is a free open-source ANN library that allows users to create multi-layer ANNs [24]. This library allows users to create ANNs quickly by including all of the calculations and processing in C files created by the authors. The user only needs to call the required functions to set the number of middle layers, the number of neurons in each middle layer, the desired error, and the separate files that contain the training and testing data sets.

4.2 Design

The overall functionality of the project can be broken down into several parts. The first part is to determine information about the network, namely the IP address and port ranges and the number of firewall rules. The second part will be the development of the firewall. Because this is only a simulation, a simulated firewall was developed with the same basic packet filtering functionalities as an actual firewall. The final part to be considered in the design is the ANN itself.

Due to the nature of the values that are used in the calculations that occur in an ANN, standard IP addresses cannot be used in the training and testing data sets. For the purpose of the simulation, the network in question will be a Local Area Network (LAN), which will allow the IP addresses used in the generation of the training and testing data sets to take the form of standard positive integers. As with a standard LAN, however, the values of these integers are limited to those between 1 and 254. Because port numbers already take the form of positive integers, no assumptions have been made regarding them.

The user will be allowed to select the number of firewall rules to be generated. However, to observe the behavior of both the simulated firewall and the ANN, the actual rules themselves will be randomly generated. As with actual firewalls, the possibility of have the *any,* or *, option in the rules has been implemented, represented by 0.

Once information about the network has been determined, the training and testing data sets will be generated. The number of “packets” in each set of data is manually determined. However, the ratio of training:testing packets is 80:20 for testing and analysis, which is the same ratio used by [2] and [6]. Typically, training sets are significantly larger than testing sets in order to allow the ANN to learn about a system’s behavior as much as possible before being introduced to new data in the testing set.

The data sets consist of five input parameters and one output value. The five input parameters considered are the source IP address, the destination IP address, the source port number, the destination port number, and the protocol used. For simplicity, only two protocols are taken into consideration: TCP and UDP. The output parameter is the action that the firewall has taken given the input parameters. Typically, this has two possible values: DENY or ACCEPT. Along with the training and testing data sets, the firewall rules that are generated will also be saved to a text file to be used by the simulated firewall. A flowchart of the packet generation phase can be seen in Figure 6.

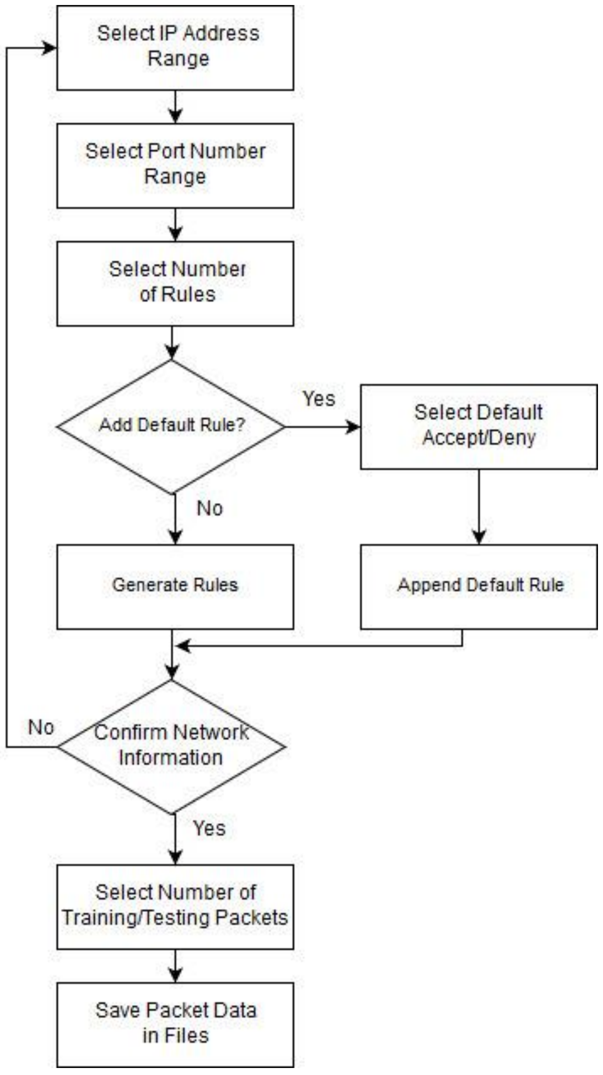


Figure 6: Flowchart of the packet generation phase

The data sets are generated in two ways, both using the network parameters selected by the user. For the training examples, the data is generated purely based on the firewall rules. This is because the training data set must include results and behavior that are *already known*. The testing set, however, includes data that may be similar to the firewall rules or *new* data that may not match any rule but still fits within the network parameters. The purpose of the testing set is to observe either system's behavior when introduced to new data.

For testing the firewall, a second testing data set has been generated that excludes the output value found in the other data sets. This is because the simulated firewall will check the rules that were generated to see if there are any matches with the current "packet" being checked and will select the appropriate action. The presence of a default rule will be taken into consideration while doing this check. However, if no default rule is found, and a packet was not found to match any of the existing rules, then the packet will be accepted by default. A flowchart depicting the simulated firewall that was designed and developed can be seen in Figure 7.

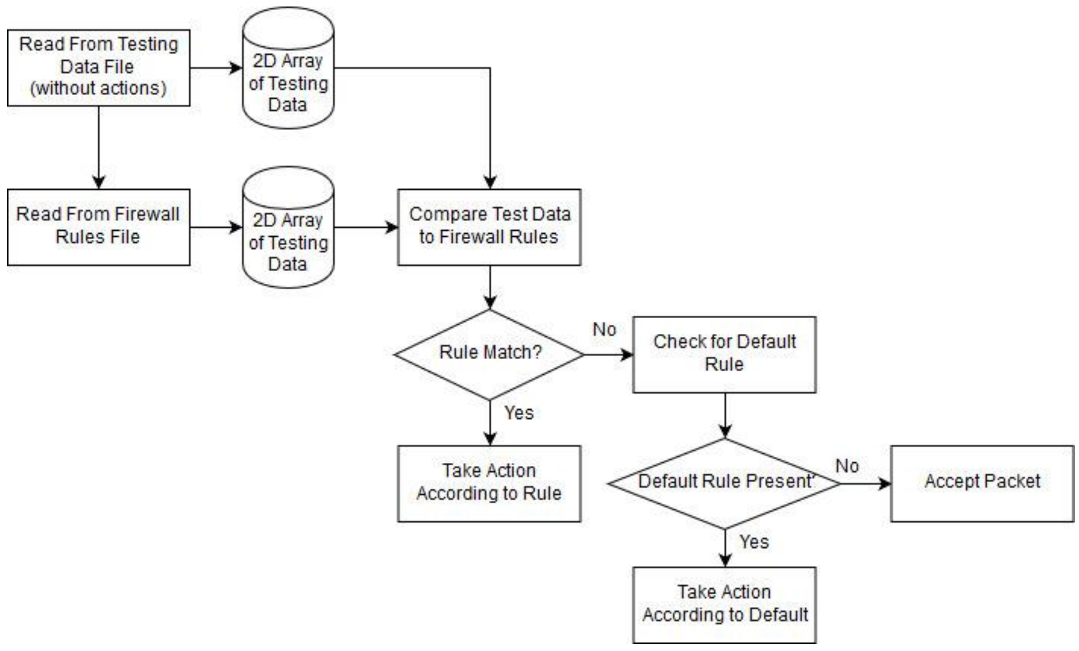
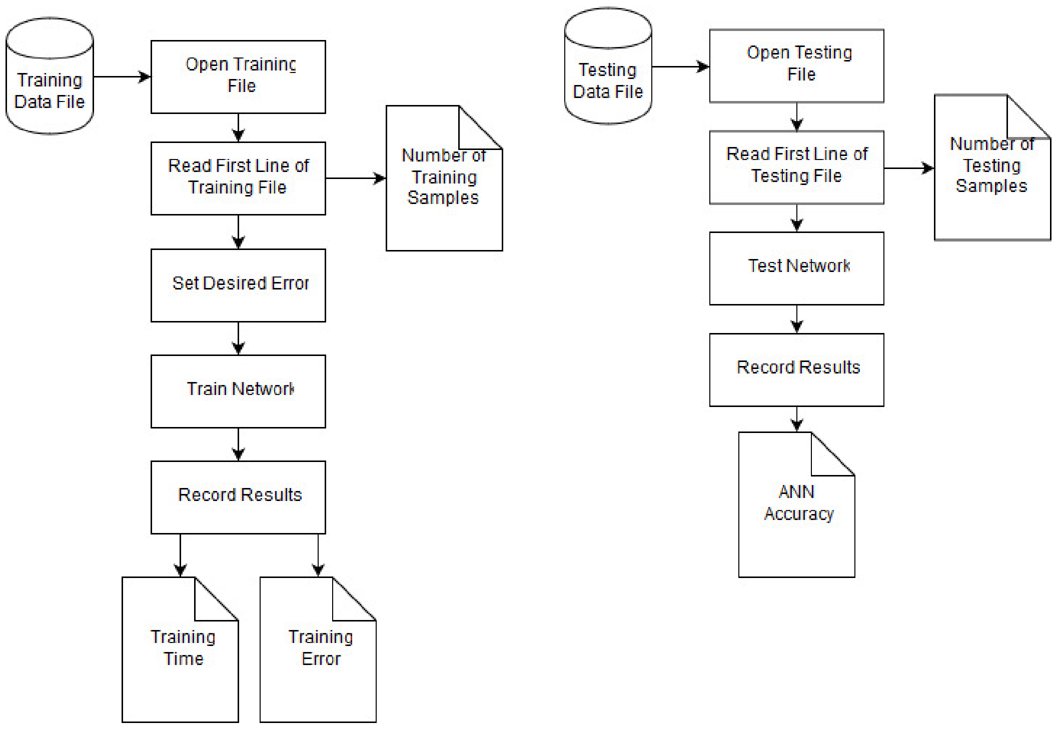

Read From Testing Data File (without actions)
2D Array of Testing Data
Read From Firewall Rules File
2D Array of Testing Data
Compare Test Data to Firewall Rules
Rule Match?
No
Yes
Check for Default Rule
Take Action According to Rule
Default Rule Present?
No
Yes
Accept Packet
Take Action According to Default

Figure 7: Flowchart of the simulated firewall

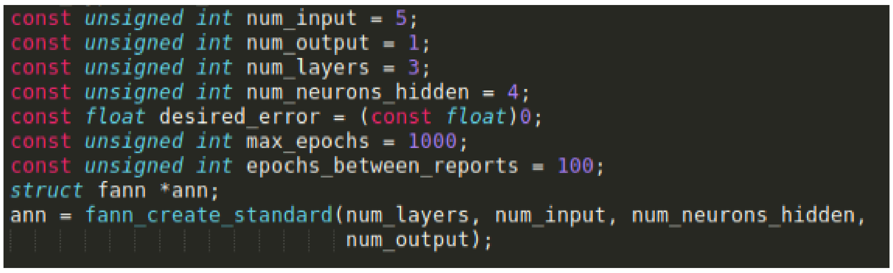
The ANN will use both the training set and the testing set containing the actions taken. It will first train itself using the training set before testing itself with the testing set. This is done to determine how well the ANN can classify packets, based on how many actions it was able to successfully assign given certain input data patterns. For packets that are considered "new" data, such as those that did not appear to follow any pattern found in the training examples, the ANN will make an "educated guess" based on the calculations that occur. The flow of the ANN's actions can be seen in Figure 8.



(a) (b)

Figure 8: Flowcharts of the ANN for the (a) training and (b) testing phases

In order for the ANN to function, certain parameters will need to be set. As previously mentioned in Section 4.1.2, the FANN library for C was used to build and run the ANN. The library allows users to select the number of inputs, outputs, and hidden layers, as well as the number of neurons in each hidden layer. The library also allows users to set a desired error and learning rate, and the algorithm used to set the weights during the training phase. Figure 9 depicts the way this is done in the code.



```
const unsigned int num_input = 5;
const unsigned int num_output = 1;
const unsigned int num_layers = 3;
const unsigned int num_neurons_hidden = 4;
const float desired_error = (const float)0;
const unsigned int max_epochs = 1000;
const unsigned int epochs_between_reports = 100;
struct fann *ann;
ann = fann_create_standard(num_layers, num_input, num_neurons_hidden,
                           num_output);
```

Figure 9: The initialization of the ANN parameters

For the ANN used in this thesis, five inputs and one output were set, and a single hidden layer with four neurons was used. Pre-tests were conducted to determine the learning rate to be used by the ANN. After several tests were done with values ranging from 0.35 to 0.8, the ideal learning rate value was found to be 0.5. This was determined by how quickly the training error was calculated, how often the training phase was cut short due to reaching the desired error value, and the final training error value.

The algorithm used to calculate and adjust the weights for the ANN is called the Nguyen-Widrow algorithm. This algorithm generates weights and bias values such that “he active regions of the layers neurons will be distributed approximately evenly over the input space” [25]. Small values are initially selected, but then are adjusted as the ANN is being trained.

Chapter 5: Testing and Results

When testing the simulation, several things need to be noted down in order to determine the performance of the ANN. Firstly, the percentage of correctly classified packets for both the ANN and the simulated firewall need to be noted down for comparison. Because the ANN's performance improves or degrades depending on how well it was trained, this percentage may fluctuate with certain amounts of data. The error of the ANN as it is training with certain amounts of training examples also need to be observed.

Another factor in the results of testing this system will be the number of rules that were generated. For this reason, the tests will occur with varying numbers of rules as well. For each set of rules, there will be a default rule that will determine the fate of packets that do not match any rule.

Table 1 contains the amounts of training and testing packets:

Table 1: The amount of training and testing examples

Training	Testing
100	25
1000	250
5000	1250
10000	2500
25000	6250
50000	12500
100000	25000
150000	37500
200000	50000

Table 1: The amount of training and testing examples (continued)

Training	Testing
300000	75000
400000	100000
500000	125000
600000	150000
700000	175000
800000	200000
900000	225000
1000000	250000

The network information used for testing will consist of an IP address range from 1 to 7 and a port number range from 2 to 263. This means that there is a total of 5,788,104 possible combinations for the samples used in this experiment. The testing phase of this thesis will iterate thrice, each iteration with an increasing number of generated rules: 4, 9, and 15.

Along with testing the ANN's behavior, the percentage of packets that were matched with a rule in the filter was also computed. This percentage was calculated with the following formula:

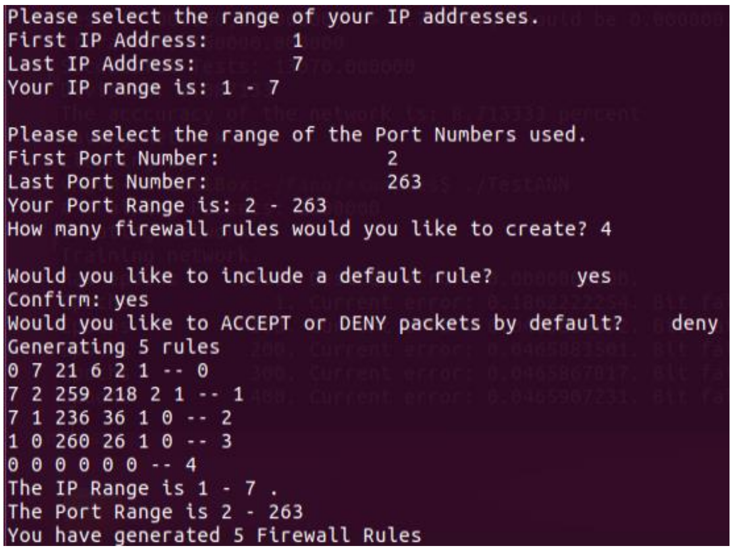
$$\% \; of \; Matched \; Packets = \frac{(\# \; of \; Packets \; Allowed \; by \; Rule) + (\# \; of \; Packets \; Denied \; by \; Rule)}{(Total \; \# \; of \; Packets)} * 100 \qquad (16)$$

In this case, packets that did not match any rule, and were thus subjected to the default rule, are considered to be incorrectly filtered. This is because in an actual network, some benign traffic that users may attempt to access can be blocked by the default DENY rule, and some malicious traffic may be accepted by a default ACCEPT rule.

5.1 With Default DENY Rule

5.1.1 Four (4) Rules

The generated network information can be seen in Figure 10.



```
Please select the range of your IP addresses.
First IP Address:         1
Last IP Address:          7
Your IP range is: 1 - 7

Please select the range of the Port Numbers used.
First Port Number:              2
Last Port Number:               263
Your Port Range is: 2 - 263
How many firewall rules would you like to create? 4

Would you like to include a default rule?         yes
Confirm: yes
Would you like to ACCEPT or DENY packets by default?    deny
Generating 5 rules
0 7 21 6 2 1 -- 0
7 2 259 218 2 1 -- 1
7 1 236 36 1 0 -- 2
1 0 260 26 1 0 -- 3
0 0 0 0 0 0 -- 4
The IP Range is 1 - 7 .
The Port Range is 2 - 263
You have generated 5 Firewall Rules
```

Figure 10: The randomly generated firewall rules

Once the data files to be used for training and testing were generated using this rule set, the ANN was trained and then tested with the generated data files, and then a testing file was used with the simulated firewall to compare the number of packets that were correctly classified. The number of samples that matched a rule in the firewall was also recorded.

The time spent training the ANN can be seen in Figure 11. As the number of training examples increased, so did the time it took to train the ANN.

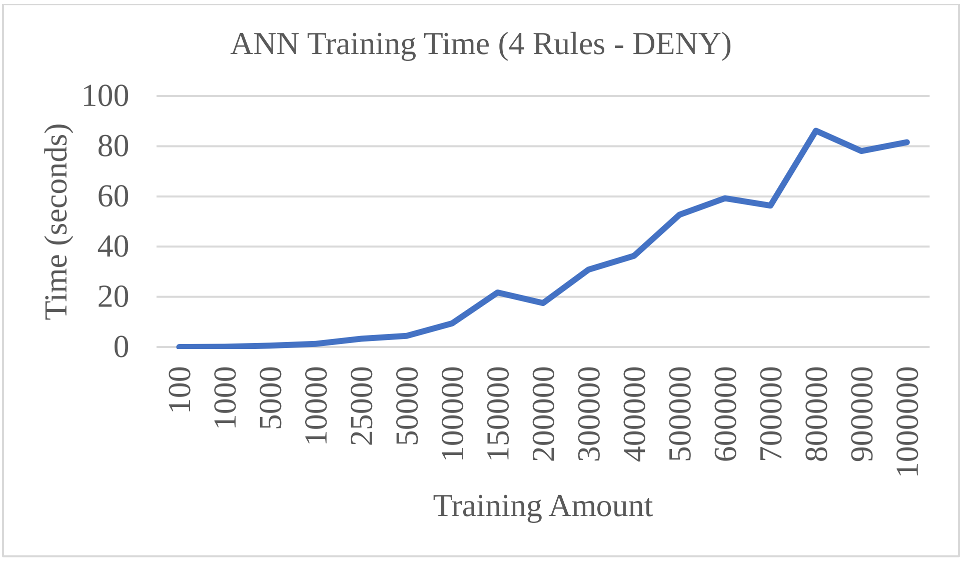


Figure 11: The training time of the ANN (4 rules)

The time spent training the network was also spent calculating the error. This is the error of the ANN's learning during the training process. A large error typically means that the ANN did not correctly learn the patterns seen in the data file, while, conversely, a smaller error denotes successful training. The training errors of the ANN with different amounts of training examples can be seen in Figure 12.

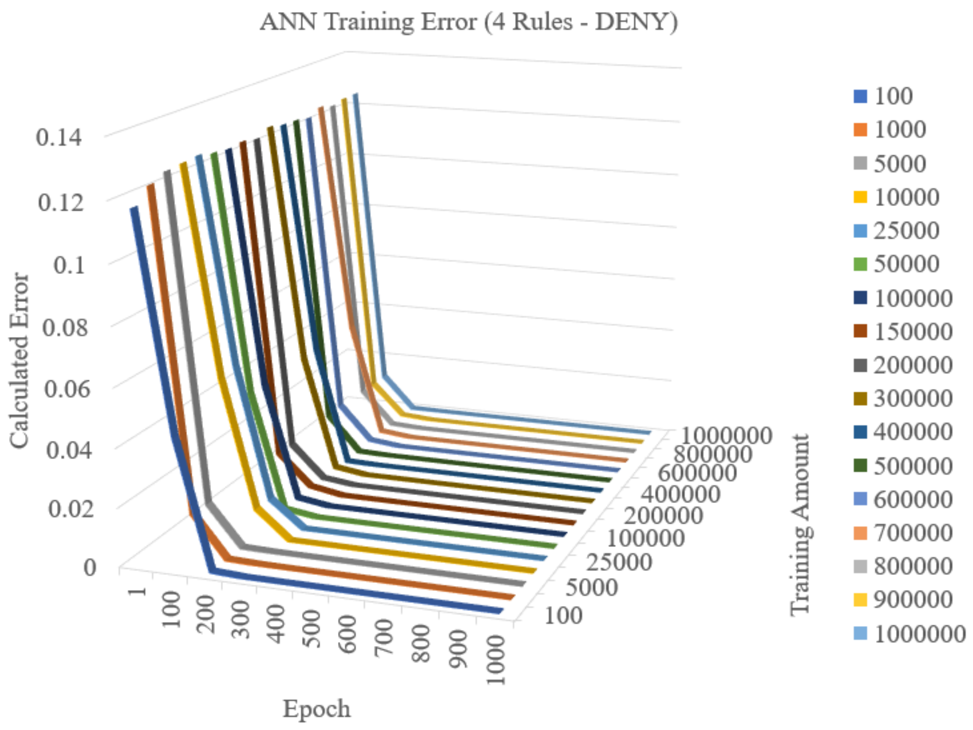


Figure 12: The ANN's error throughout the training phase

The curves in this graph depict how smoothly the ANN was able to train with the given amount of training examples. It can be noted that for smaller amounts of training examples, the curve approaches zero much later than with larger training sets. As the number of training examples increases, the curve becomes smoother, meaning that the ANN was able to learn the data patterns well. A spiky or rough curve indicates that the ANN may be having a difficult time learning the patterns found in the training set. This could be a result of weights needing to be adjusted.

Once the training of the ANN was completed, it was tested using the testing examples generated. The simulated firewall was also tested. Their accuracies during filtering can be observed in Figure 13.

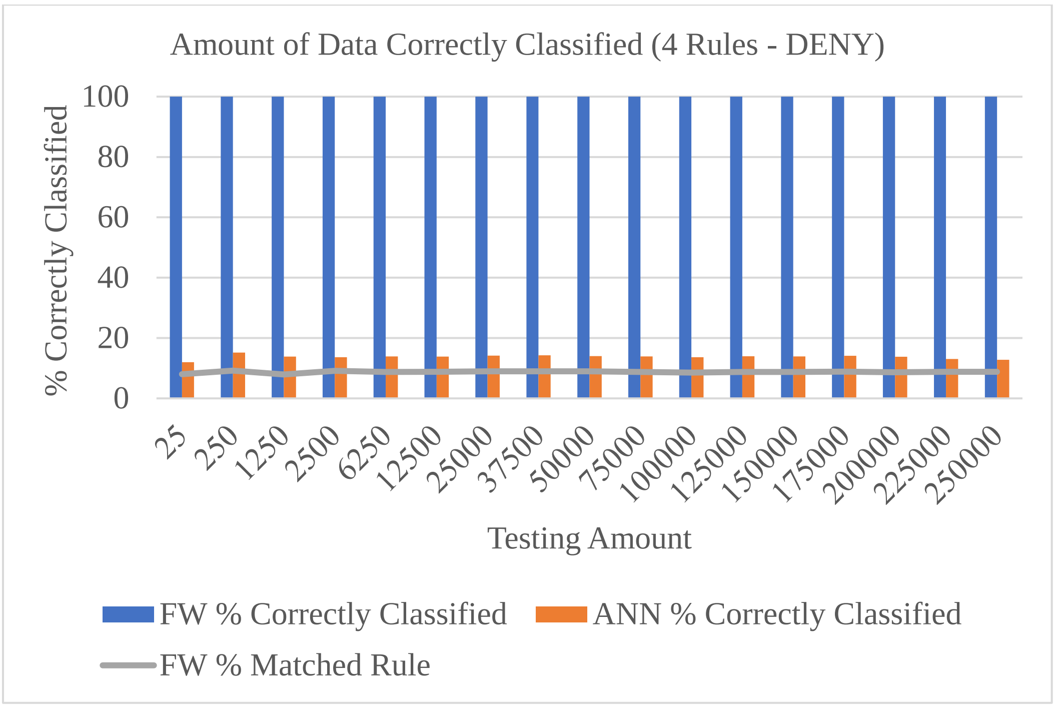
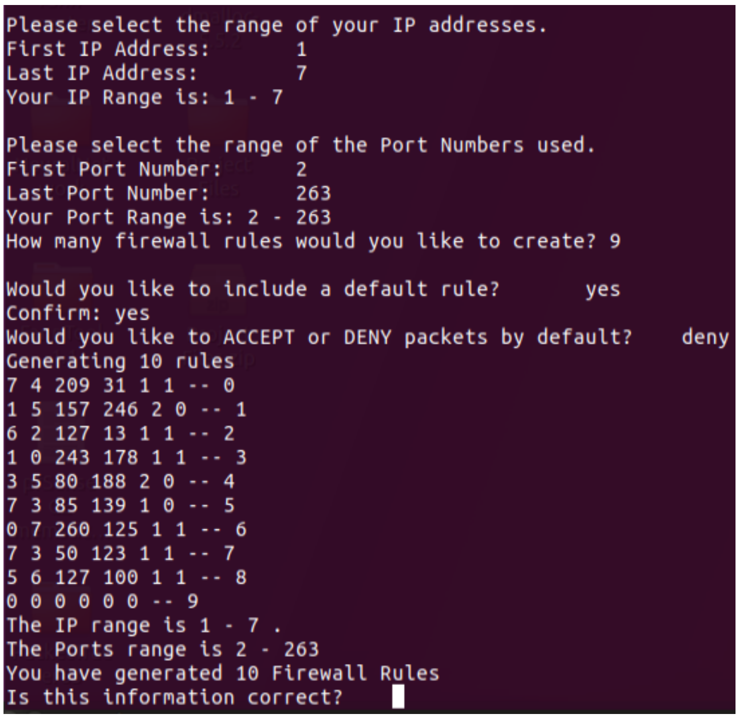


Figure 13: Comparison of the percentage of correctly classified samples

As can be seen from the figure, the percentage of packets that were correctly classified by the simulated firewall’s remained constant at 100%. This is because the firewall has set policies with which packets are filtered. Therefore, the firewall’s judgement is always correct. The ANN, on the other hand, had a low percentage of correctly classified packets. This percentage, however, was close to the percentage of packets that matched a firewall rule while testing the firewall. Due to the relatively small number of rules experimented with given the size of the network, however, these results are statistically insignificant.

5.1.2 Nine (9) Rules

Figure 14 displays the rules that were generated when the system was testing with 9 rules.



```
Please select the range of your IP addresses.
First IP Address:        1
Last IP Address:         7
Your IP Range is: 1 - 7

Please select the range of the Port Numbers used.
First Port Number:       2
Last Port Number:        263
Your Port Range is: 2 - 263
How many firewall rules would you like to create? 9

Would you like to include a default rule?       yes
Confirm: yes
Would you like to ACCEPT or DENY packets by default?    deny
Generating 10 rules
7 4 209 31 1 1 -- 0
1 5 157 246 2 0 -- 1
6 2 127 13 1 1 -- 2
1 0 243 178 1 1 -- 3
3 5 80 188 2 0 -- 4
7 3 85 139 1 0 -- 5
0 7 260 125 1 1 -- 6
7 3 50 123 1 1 -- 7
5 6 127 100 1 1 -- 8
0 0 0 0 0 0 -- 9
The IP range is 1 - 7 .
The Ports range is 2 - 263
You have generated 10 Firewall Rules
Is this information correct?
```

Figure 14: Network information with 9 rules

The time spent training the ANN with the packets generated from the 9 rules seen in Figure 13 can be seen in Figure 15.

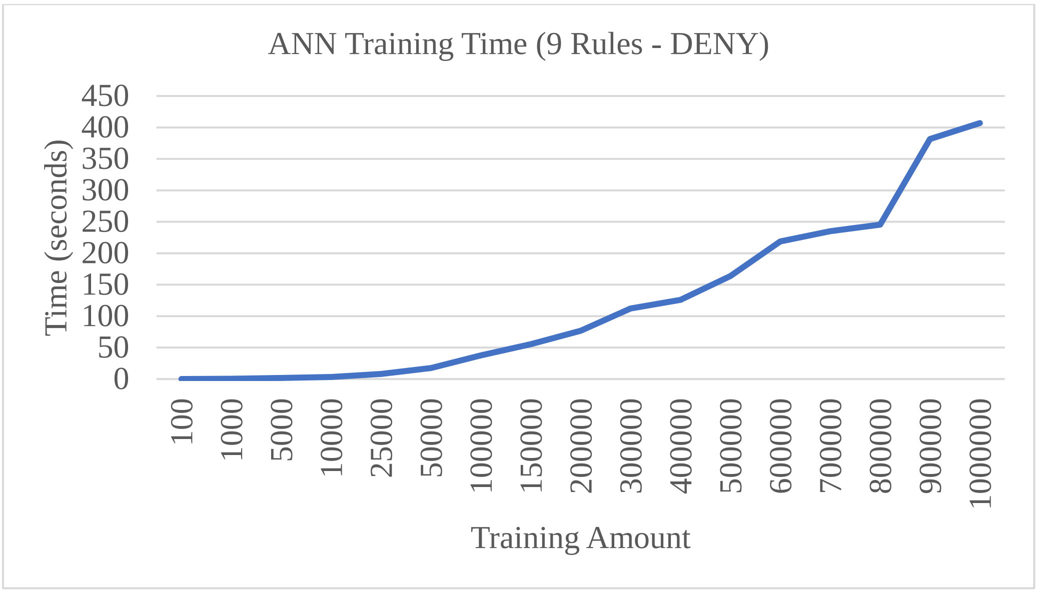


Figure 15: The amount of time spent training the ANN

During these training phases, the training errors generated were calculated and can be observed in Figure 16.

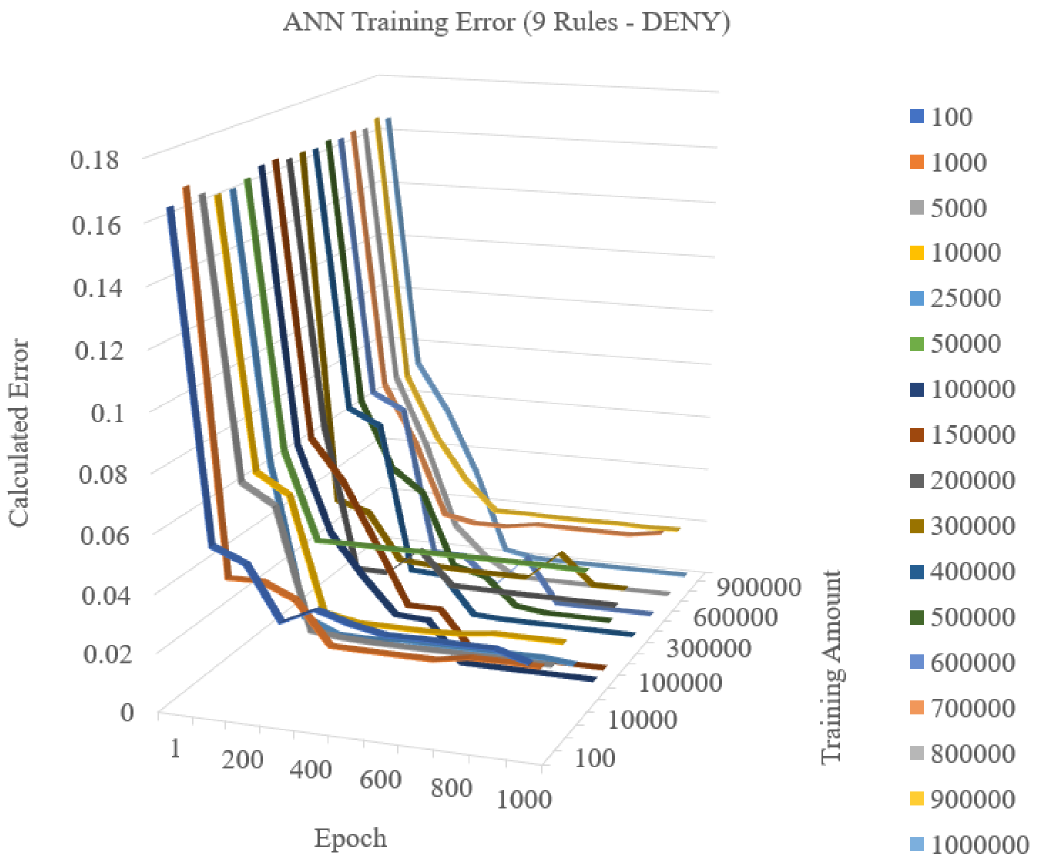


Figure 16: The ANN's error throughout the training phase

The graphed lines in this figure are notably rougher than the training errors in Figure 11. This is due to the increased number of rules. The ANN is learning more patterns in this iteration of the experiment, which may cause some miscalculation during the training phase.

With the packets that were generated with these rules, as well as some unknown, random packets that have been generated for testing purposes, the accuracies at different amounts of testing packets were observed and organized in Figure 17.

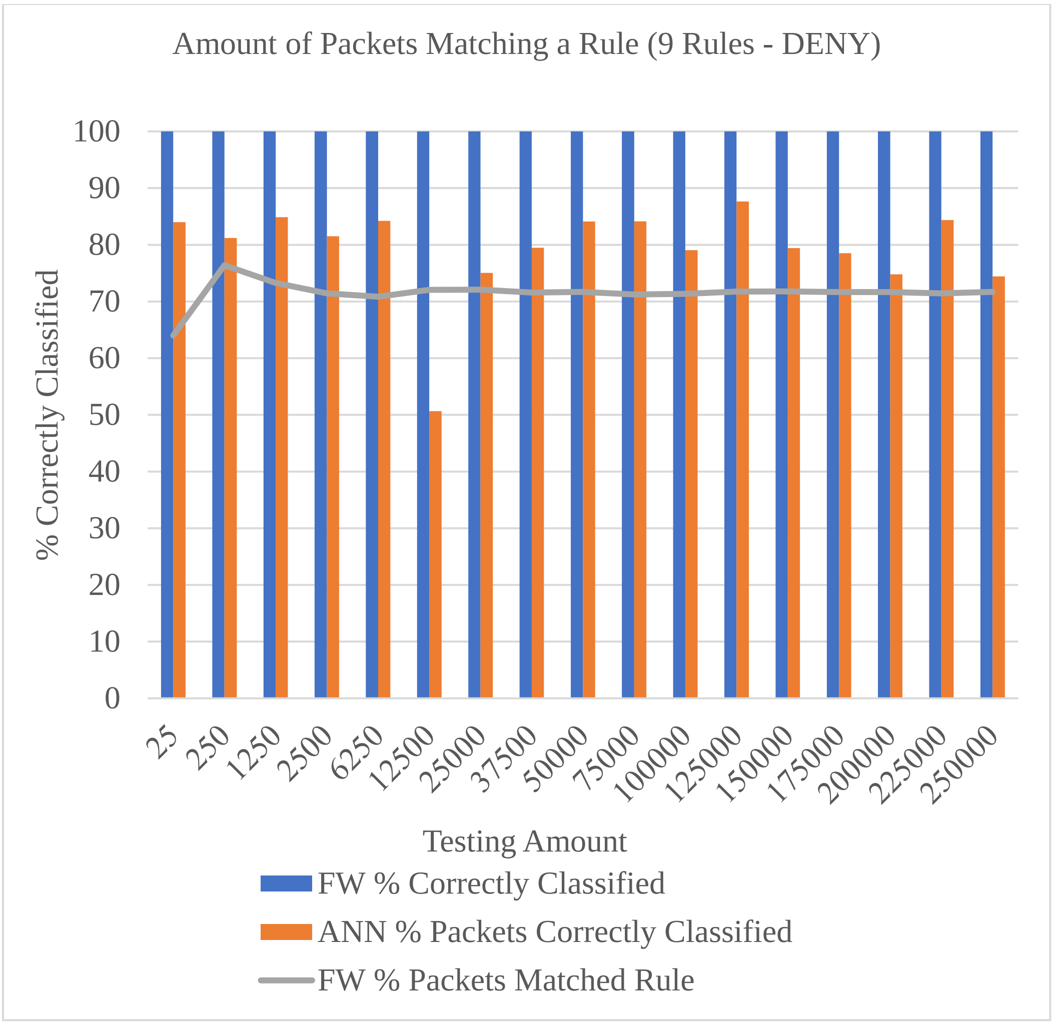
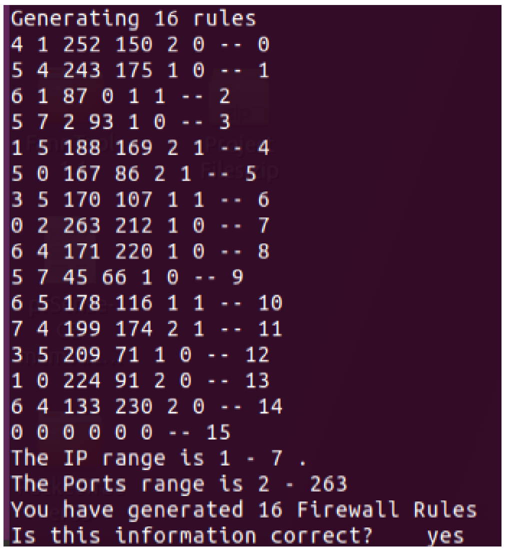


Figure 17: Comparison of the percentage of correctly classified samples.

5.1.3 Fifteen (15) Rules

The 15 rules generated for the final testing phase of the system can be seen in Figure 18.



```
Generating 16 rules
4 1 252 150 2 0 -- 0
5 4 243 175 1 0 -- 1
6 1 87 0 1 1 -- 2
5 7 2 93 1 0 -- 3
1 5 188 169 2 1 -- 4
5 0 167 86 2 1 -- 5
3 5 170 107 1 1 -- 6
0 2 263 212 1 0 -- 7
6 4 171 220 1 0 -- 8
5 7 45 66 1 0 -- 9
6 5 178 116 1 1 -- 10
7 4 199 174 2 1 -- 11
3 5 209 71 1 0 -- 12
1 0 224 91 2 0 -- 13
6 4 133 230 2 0 -- 14
0 0 0 0 0 0 -- 15
The IP range is 1 - 7 .
The Ports range is 2 - 263
You have generated 16 Firewall Rules
Is this information correct?    yes
```

Figure 18: Network information with 15 rules

The time it took to train the ANN with the training examples generated from this set of rules can be seen in Figure 19.

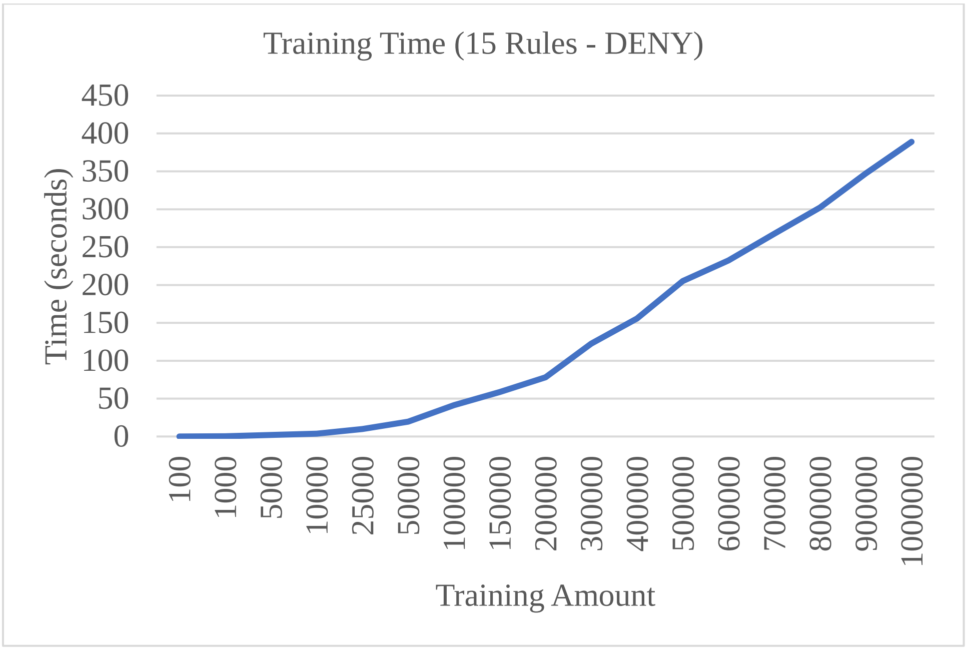


Figure 19: The training time of the ANN with different amounts of examples

The errors with different amounts of data during the training phase can be seen in Figure 20.

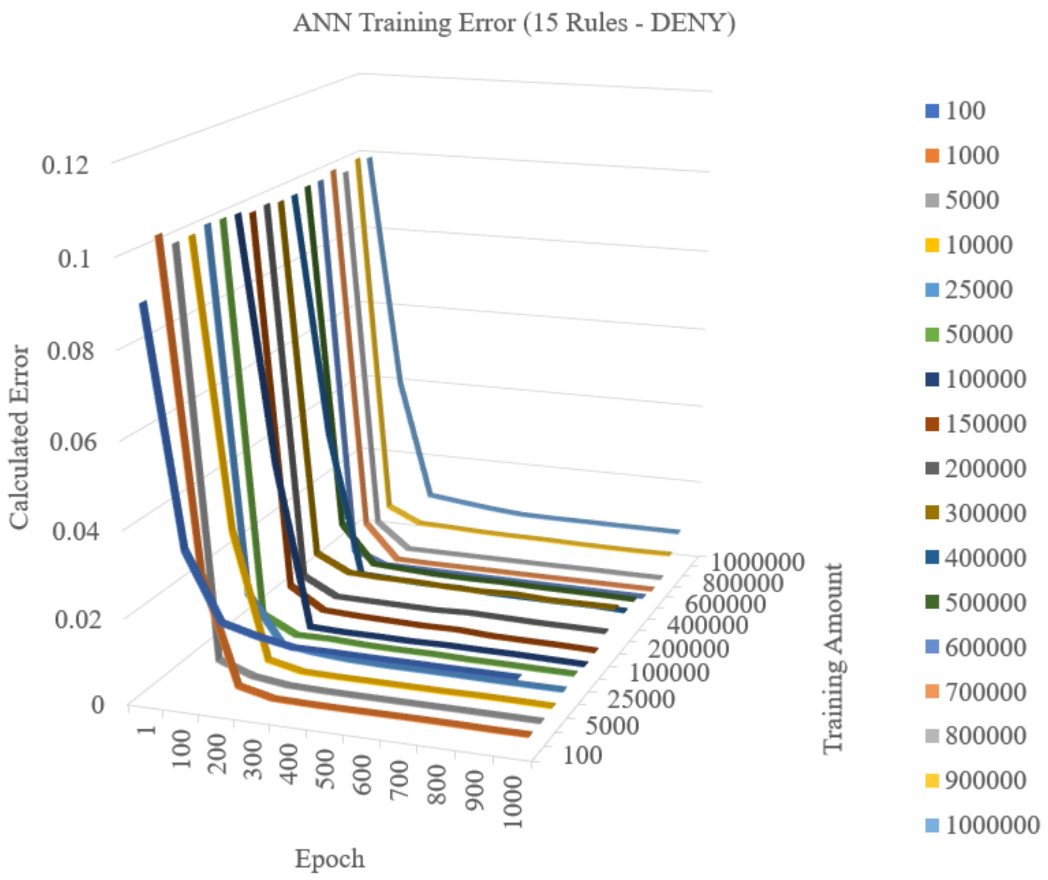


Figure 20: The training errors generated

Once training was completed, both the ANN and the simulated firewall were fed the testing samples to determine the amount of correctly classified packets. The results of the testing phase can be seen in Figure 21.

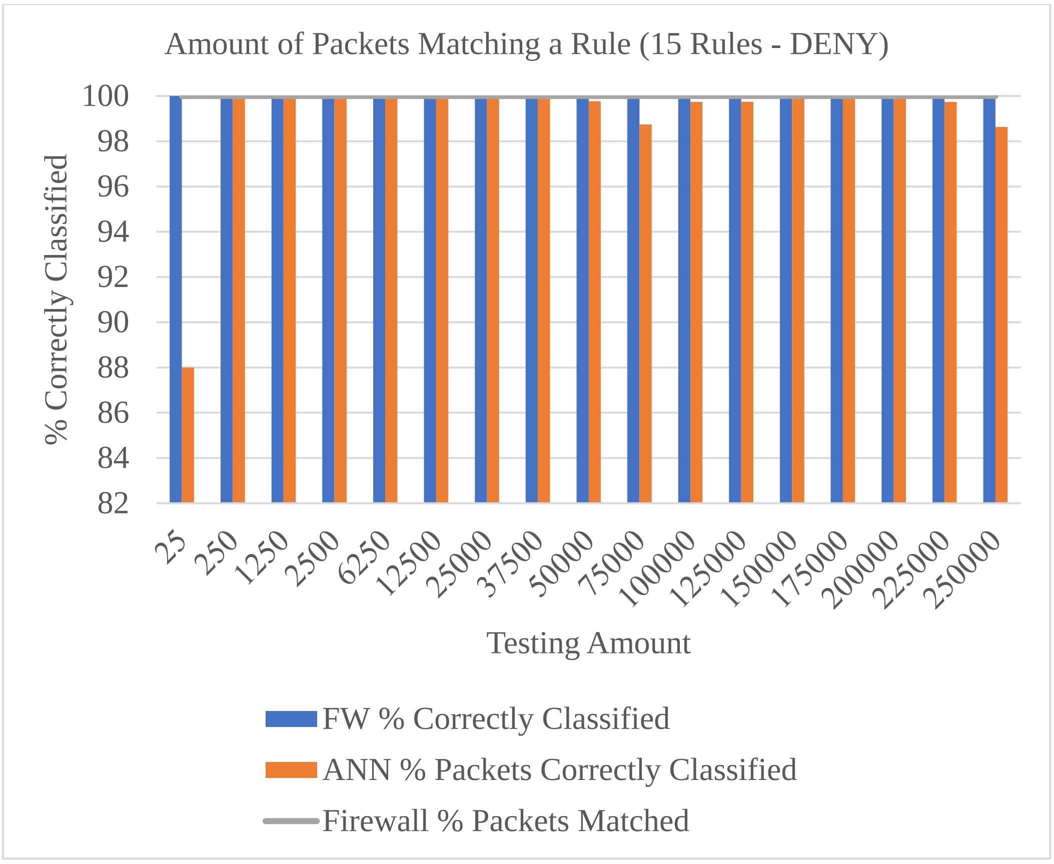


Figure 21: Comparison of the percentage of correctly classified samples

From the figure, it can be observed that the firewall was able to match all test samples to a rule. This is due to the fact that there are enough rules relative to the network size to be able to match one.

5.2 With Default ACCEPT

The same tests were completed with the same sets of rules. However, in this case, the behavior of the ANN will be observed when new packets are subjected to a default rule allowing any packets that don’t match any rules into the network. Examples of such networks can be found in public places, such as malls or café`s.

5.2.1 Four (4) Rules

The time it took to train the ANN with the training examples generated with the 4 rules in Figure 9 can be seen in Figure 22.

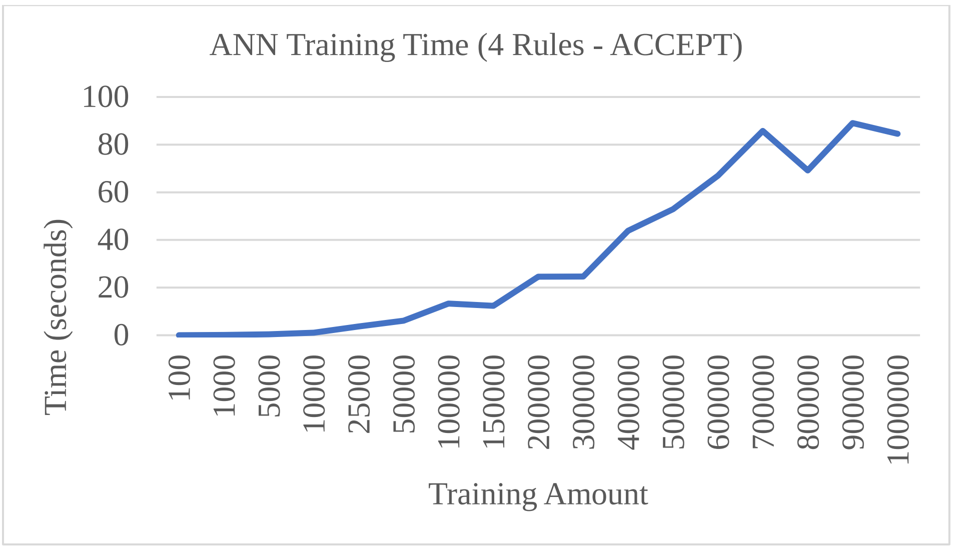


Figure 22: The training times of the ANN

The training errors generated by the ANN during this phase can be seen in Figure 23.

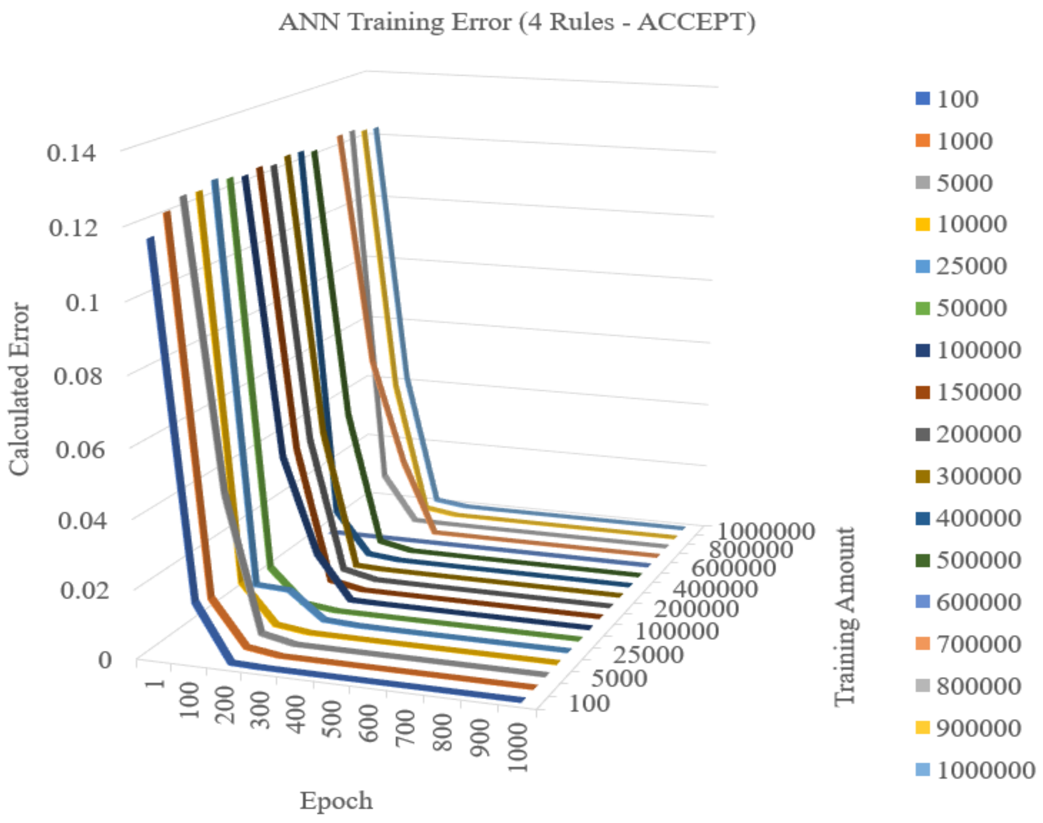


Figure 23: The errors calculated during the training phase

Once the training phase has been completed, the ANN and the simulated firewall were given the test samples. The results of the testing phase can be seen in Figure 24.

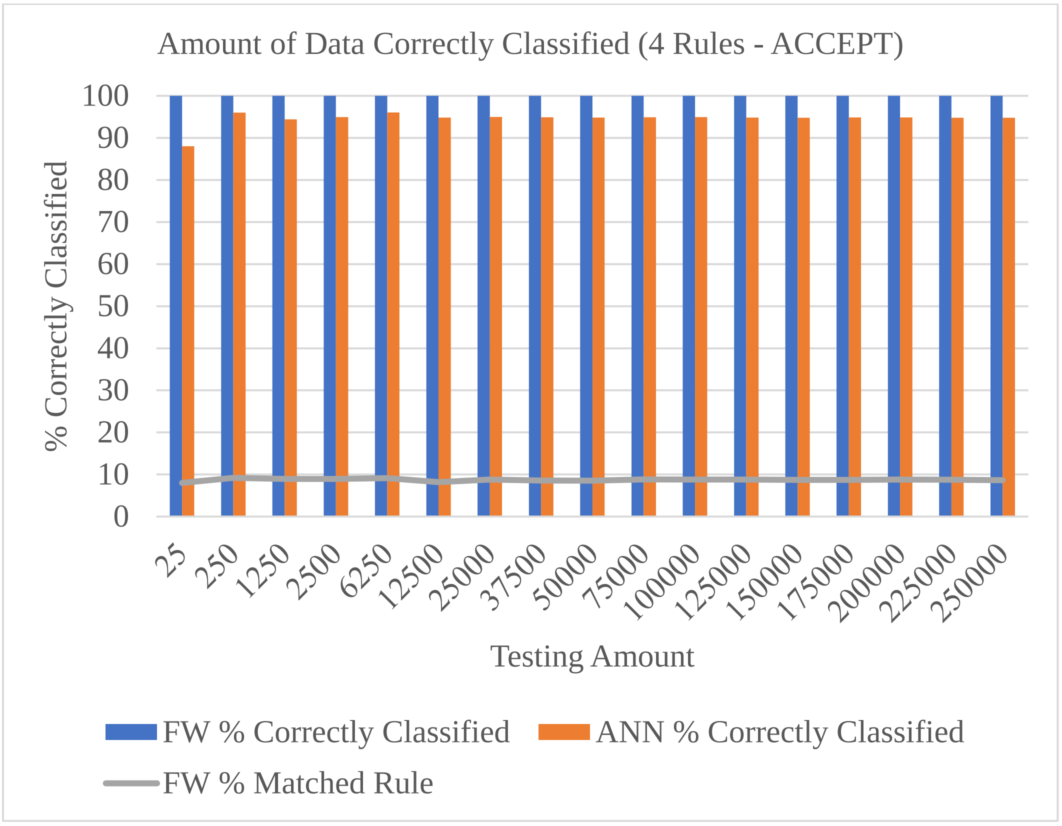


Figure 24: Comparison of the percentage of correctly classified samples

5.2.2 Nine (9) Rules

The training times at varying amounts of training examples generated by the 9 rules in Figure 13 can be seen in Figure 25.

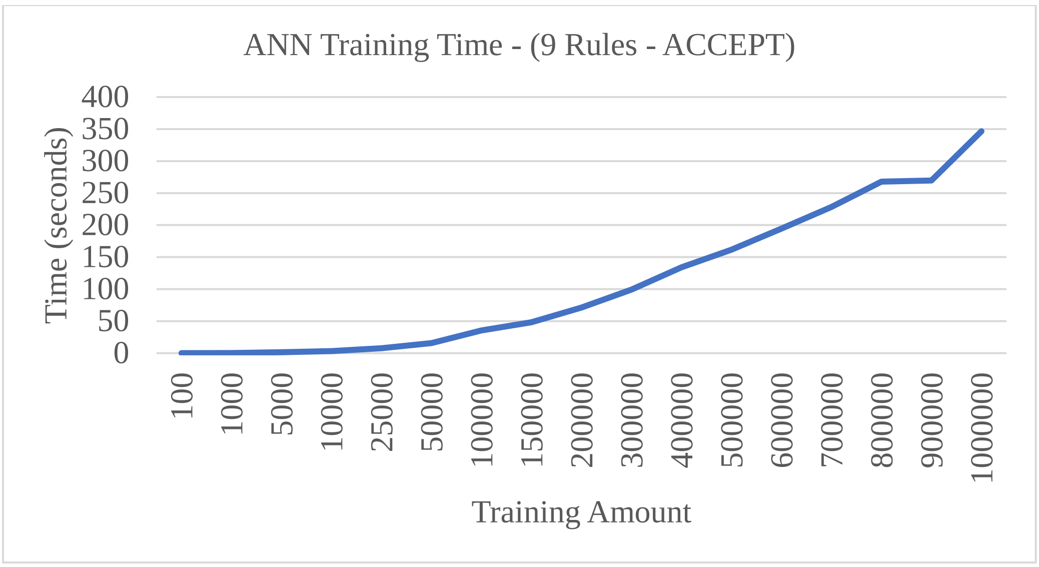


Figure 25: The training times with varying amounts of training examples

The training errors calculated during this phase can be seen in Figure 26.

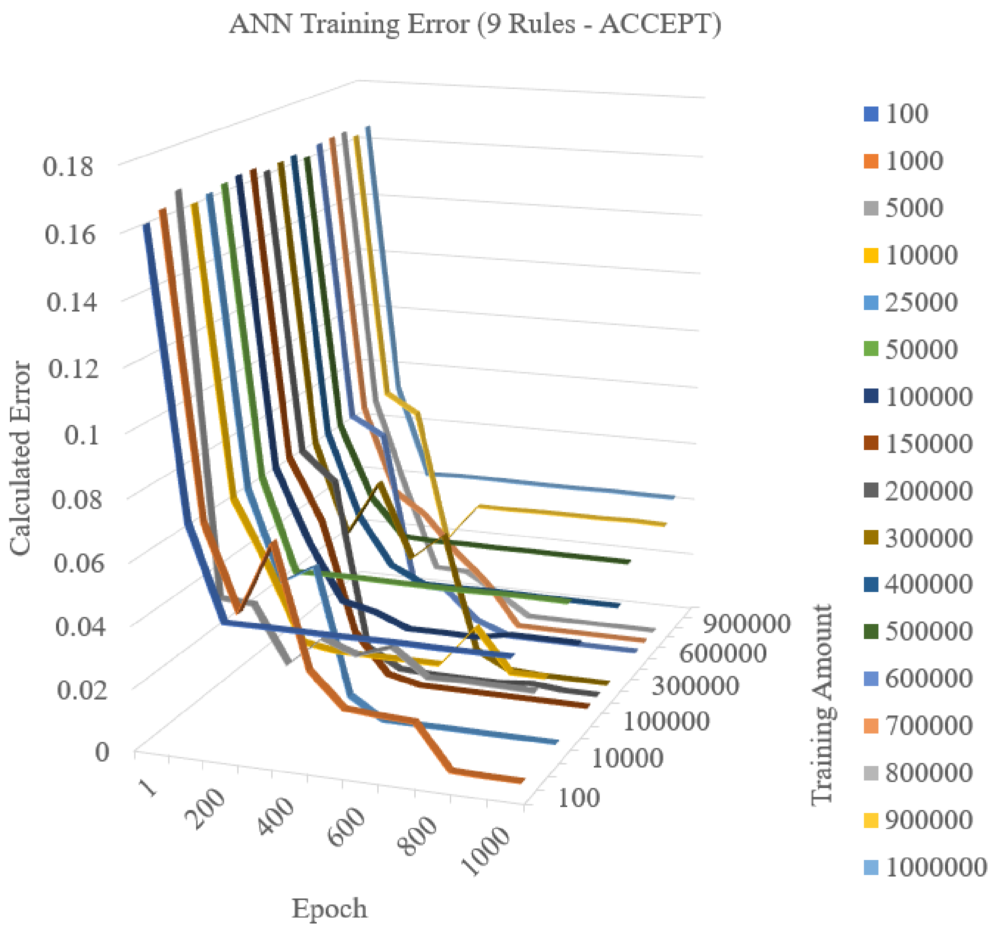


Figure 26: The calculated training errors during the ANN's training phase

In this figure, the same spikes in the training curves as in Figure 15 can be noted. This is due to the same reason that was previously mentioned; the ANN is given more patterns to learn and will need to adjust and learn each one.

Once the training phase was completed, the ANN and the simulated firewall were tested using testing examples generated from the 9 rules. The results can be observed in Figure 27.

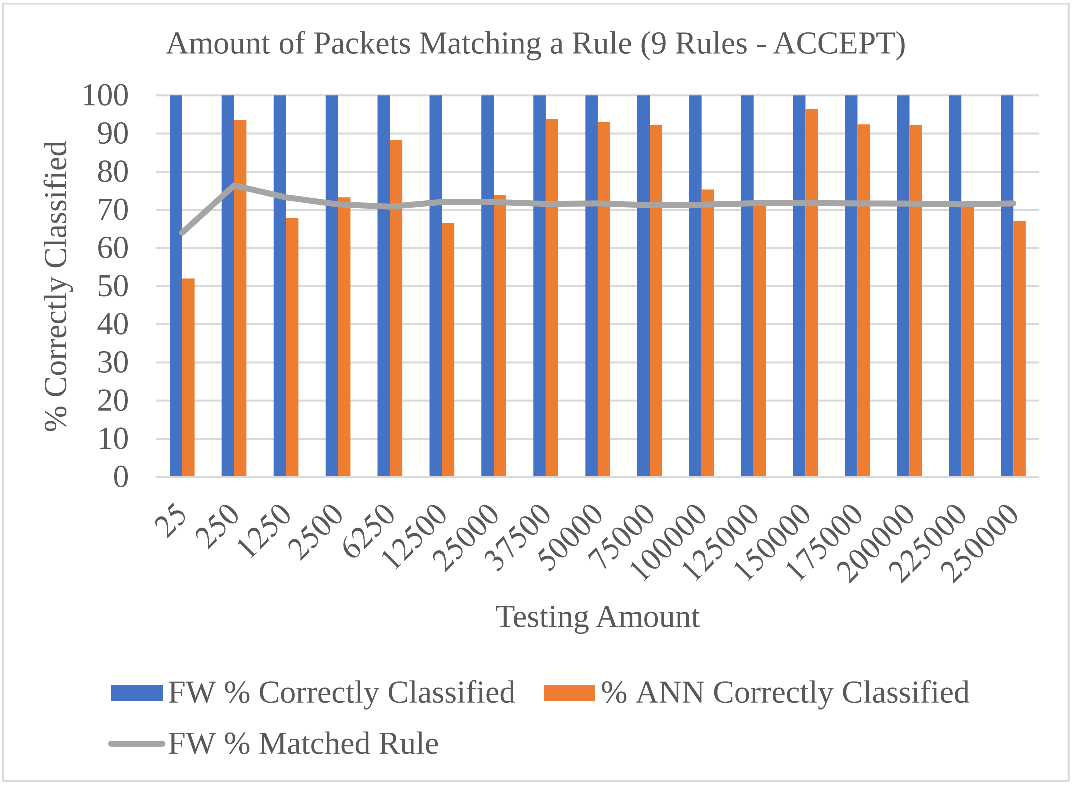


Figure 27: Comparison of the percentage of correctly classified samples

The percentage of packets correctly classified by the ANN is closer in value than what was observed in Figure 23.

5.2.3 Fifteen (15) Rules

During the final phase of the system's experiments using a default accept rule, the same 15 rules as shown in Figure 17 were used to test the ANN and simulated firewall. The time it took to train the ANN with different amounts of training examples generated from these rules can be seen in Figure 28.

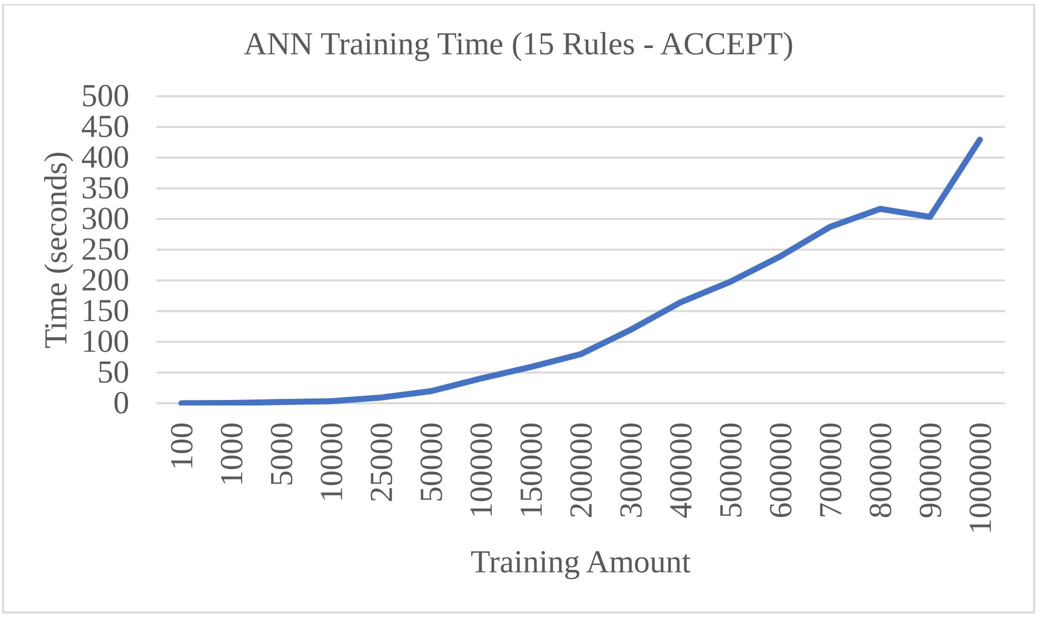


Figure 28: The training time of the ANN

The training errors that were calculated with varying amounts of training examples during this phase can be seen in Figure 29.

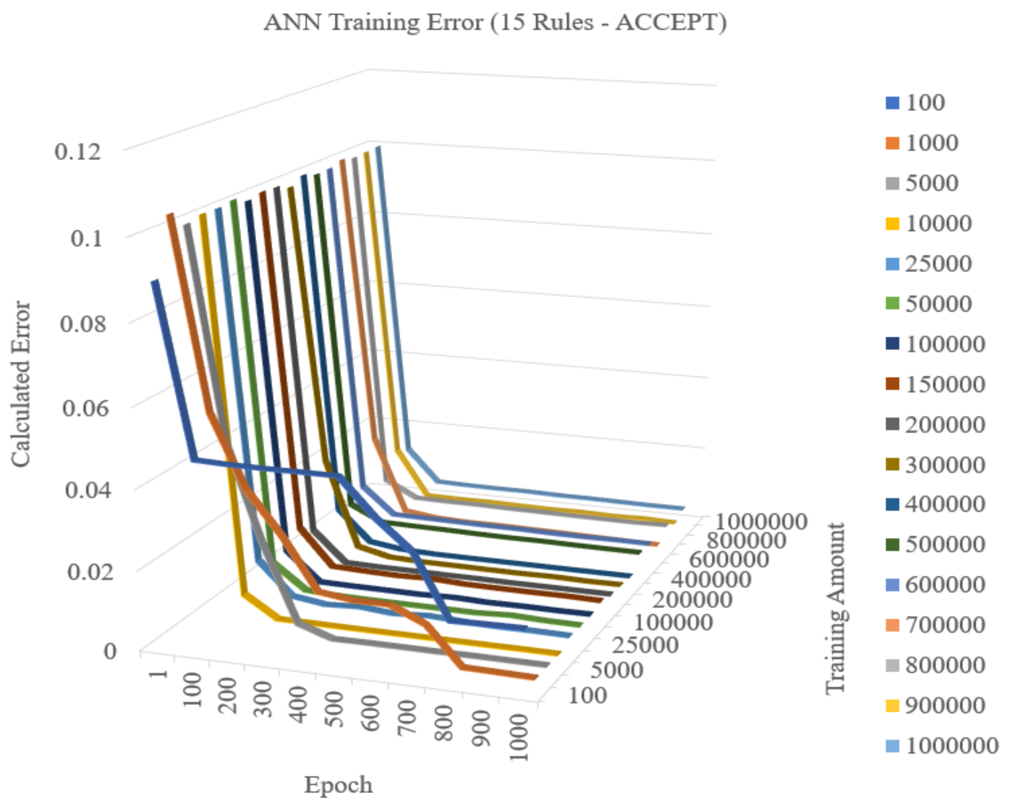


Figure 29: The training errors generated during the ANN's training phase

Once the training phase has been completed, the ANN and the simulated firewall were tested while taking into considering the default accept rule. The results of the testing phase can be seen in Figure 30.

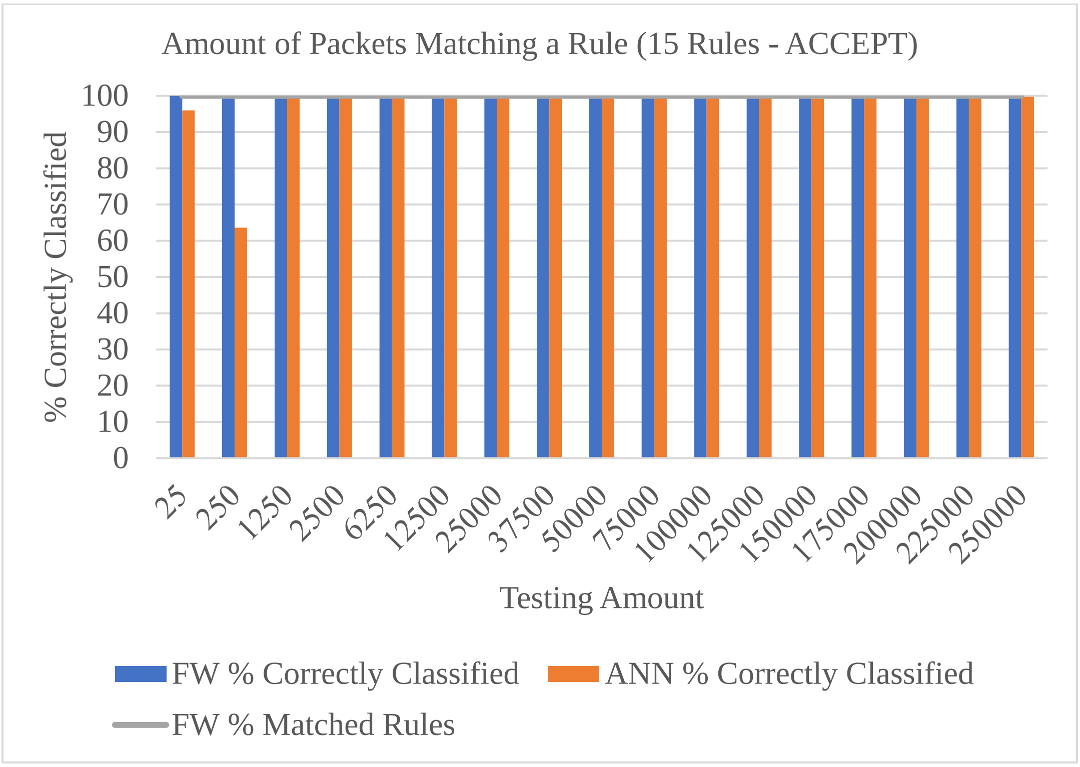


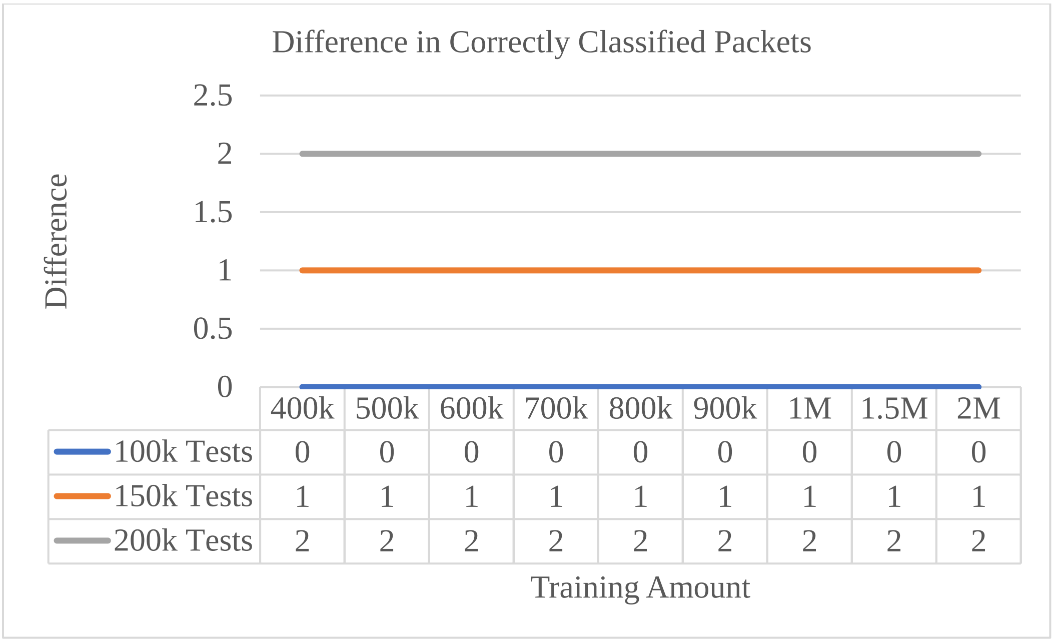
Figure 30: Comparison of the percentage of correctly classified samples

5.3 Uncertainties

While running the tests with the ANN, a difference in the number of examples that were successfully filtered was noticed between the simulated firewall and the ANN. These amounts were recorded and observed. The example packets involved in these differences were named *Uncertainties*. These examples represented the packets that may not have been correctly classified by the ANN. Such a packet may be a benign one that was wrongfully denied entrance to the network, or a malicious packet that was wrongfully accepted.

5.3.1 Four (4) Rules

While the tests were run with 4 rules, the difference in successfully classified examples between the amount firewall and the ANN. A graph of these values can be seen in Figure 31.



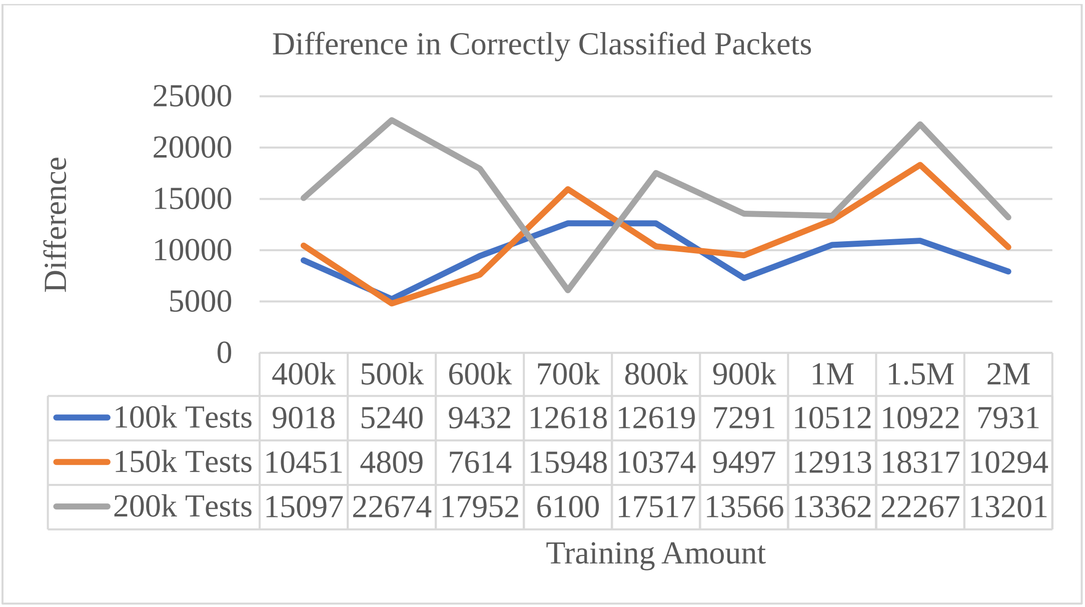
	400k	500k	600k	700k	800k	900k	1M	1.5M	2M
100k Tests	0	0	0	0	0	0	0	0	0
150k Tests	1	1	1	1	1	1	1	1	1
200k Tests	2	2	2	2	2	2	2	2	2

Figure 31: The differences in successful classifications (4 rules)

From this graph, there is no real trend that can be noticed with increasing training sample amounts. This is in tandem with what was previously said: that the results observed while using 4 rules are statistically insignificant. For the amount of possible packet combinations in the experiments due to the size of the network chosen, 4 rules is too few and will not allow packets to be filtered correctly by both the ANN and the firewall. The ANN will face difficulties because there aren't enough patterns learned for the wide variety of input sequences in the data, and the firewall will subject many of the packets to the default rule.

5.3.2 Nine (9) Rules

The amounts of Uncertainties with 9 rules can be seen in Figure 32.



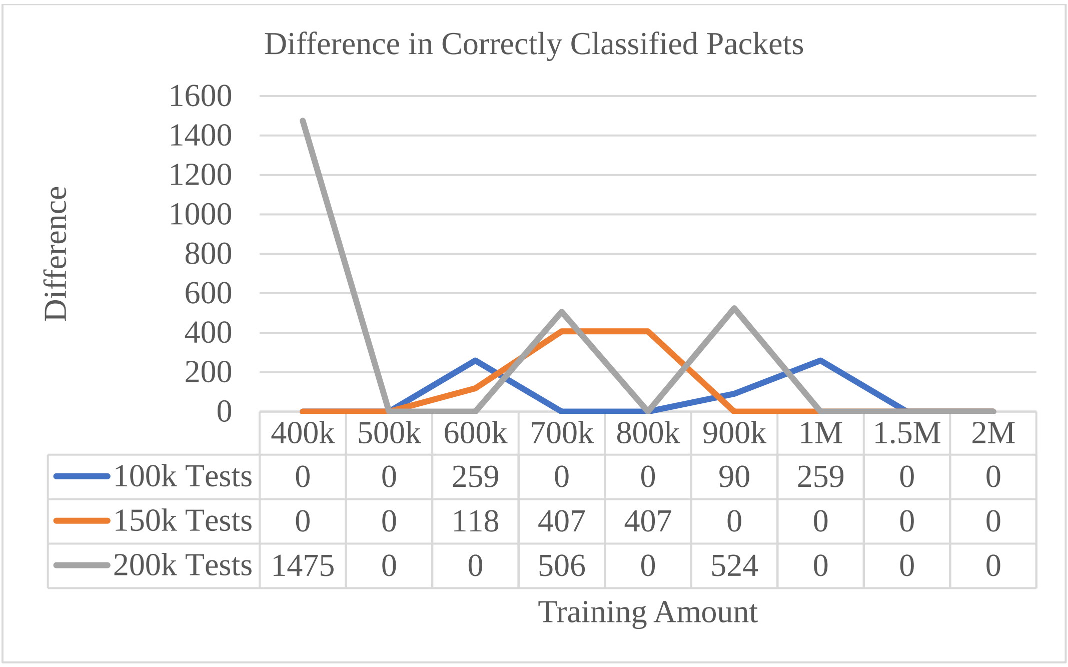
	400k	500k	600k	700k	800k	900k	1M	1.5M	2M
100k Tests	9018	5240	9432	12618	12619	7291	10512	10922	7931
150k Tests	10451	4809	7614	15948	10374	9497	12913	18317	10294
200k Tests	15097	22674	17952	6100	17517	13566	13362	22267	13201

Figure 32: The differences in successful classifications (9 rules)

Although the differences in the amount of correctly classified packets between the ANN and the firewall are greater in this case, the actual values of the differences do not differ very much between the varying number of tests. Using 9 rules shows an improvement while classifying the data, with an overall trend of decreasing over time. This shows that training the ANN more will provide better results, at the cost of taking longer. Given the amounts of test examples used during the Testing Phase and the total number of data combinations, the difference is relatively small.

5.3.3 Fifteen (15) Rules

The number of Uncertainties calculated while using 15 rules can be seen in Figure 33.



	400k	500k	600k	700k	800k	900k	1M	1.5M	2M
100k Tests	0	0	259	0	0	90	259	0	0
150k Tests	0	0	118	407	407	0	0	0	0
200k Tests	1475	0	0	506	0	524	0	0	0

Figure 33: The differences in successful classifications (15 rules)

In this case, the decrease in the differences is clearer. As seen in Sections 5.1.3 and 5.2.3, the firewall perfectly filtered all packets without the use of the default rule. The ANN is also able to closely match the firewall in the number of correctly classified samples. Given the number of testing samples used in the Testing Phase and the size of the network, the difference between the ANN and FW's amounts correctly classified packets is very small. This shows that an increase in the number of rules allows the ANN to learn more about a network and reduce the likelihood of false classification.

5.4 Observation

Several things can be noted while reviewing the results of the tests done with the ANN and the simulated firewall. The first thing to note is that, in all cases, the time it took the train the ANN increased as the number of training examples increased, as expected. This is because the ANN will have more data to work with, and thus, more calculations will need to be done to make connections between neurons.

The second thing to note is that the training error decreased over the course of the training period, regardless of the number of training examples that the ANN was encountering. As was mentioned before, the training error that is being calculated here gives an idea of how well the ANN is discovering and learning patterns that are found within the training examples. The higher the error is, the worse the ANN is learning. It can also be noted that in all cases, the training error that is calculated is significantly higher in the first epoch before taking a dive after 100 epochs. This is because the ANN does not know what the patterns in the training examples are once training begins. After 100 epochs, the ANN would have begun to learn the patterns found within the training examples, hence the sharp dive shown in the graphs.

The number of correctly classified packets between the ANN and the simulated firewall is another aspect to note. When fewer rules are involved, the difference in the number of correctly classified packets between the ANN and the firewall are great. As seen in Figure 13, the ANN was not able to correctly classify many data samples. This is because of the lack of patterns that it needs to learn. With more than 5 million data combinations as a result of the size of the network used in the experiments, using 4 rules results in statistically insignificant data. The results of the tests with 9 and 15 rules represent what the results would be in an actual network more accurately.

One factor of the results is the firewall's ability to accept ANY as an option during filtering. In this system, the ANY option was replaced by the value 0 for testing purposes. The firewall has a system for understand that if ANY appears in a firewall rule, then any value that appears in that position is acceptable and subjected to the action of choice for that rule. The ANN, on the other hand, takes data from the firewall and has no understanding of what ANY means. If packets are received that look identical except for one or two values that are to be seen as ANY by the firewall, the ANN may become confused and may not come to the same conclusion as the firewall.

Chapter 6: Discussion

6.1 Advantages

There are several advantages of using ANNs over firewalls in order to process packets for filtering. The first advantage is that the ANN’s ability to extract and analyze large amounts of data, and then store them for forecasting later on. Aside from logging filtering information, firewalls do not do this when a packet has been filtered. This leads to the second advantage of using ANNs: the ability to have nonlinear relationships with the input data. This means that ANNs are able to map values and predict output information more accurately than a linear system. The difference in mapping between a linear and nonlinear systems can be seen in Figures 1 and 4.

The lack of the need for extensive knowledge in statistics is a third advantage of using ANNs. When programming ANNs, there are many open source guides and libraries that allow anyone with an objective to create ANNs. The C library mentioned in subsection 4.2.2, FANN, is one example of such a library. Another example is Google’s TensorFlow, a Python-based library for dataflow and symbolic math.

The final, major advantage of using ANNs is the flexibility of design and development. There are several different types of ANNs that can be designed around a number of applications, depending on user needs. There are also a selection of different training models that users can implement based on the context of the data being fed to the ANN. The data itself is also flexible, in the sense that it can come from any source, whether it is a text, image, or audio file.

6.2 Disadvantages

Although ANNs have several strong advantages compared to firewalls, there are also some disadvantages that need to be taken into consideration when designing a system. The first important disadvantage to consider is the possibility of having long, variable training times. The time it takes to train an ANN varies depending on the size and amount of data that is being input into the system. For small datafiles, the training time may not be an issue. However, in systems that may deal with large amounts of data, such as large, active networks, the training time may take much longer.

This relates to the second disadvantage of using ANNs. If an ANN is to be employed in a critical, active environment, such as an organization’s network, it will need to be retrained every once in a while. If the datasets used for training are large, retraining could take time, potentially putting the network at risk. In order to be employed in the first place, the ANN’s accuracy after training and testing will need to reach a certain threshold. This means that there is also the possibility of having to train, and then later retrain, the ANN multiple times before the threshold has been achieved. Employing the ANN otherwise may be a risk on the network, since the ANN may not be completely trained and ready to face the data being communicated.

The third disadvantage of using ANNs is that there is no understanding of the data that is being processed. An ANN’s purpose is to detect patterns and return an output based on what it has learned from its training phase. There is no meaning to the data that is being fed and processed. In the case of the intended application discussed in this thesis, the ANN alone will not be enough to filter packets and assign actions appropriately. Modifications and additions to the code will need to be made in order for the network to handle the packets being filtered appropriately.

Lastly, the ANN may face difficulty filtering packets correctly in an unstable network. In this case, the nature of the traffic may not be well-defined, and the ANN may not make the appropriate decisions.

6.3 Recommendations

Because the system designed and developed in this thesis is a simulation to determine if ANNs can work alongside firewalls to improve packet filtering capabilities, the work done paves the way for future development in this topic. There are improvements and modifications that can allow the ANN to work as a packet filtering tool in a network. The first is to implement this system with an actual LAN with a few test computers and have them communicate with each other normally. In this case, the code of the ANN will need to be modified to not only determine the fate of a packet passing through a network, but to also act upon the decision made. Implementing this system with an actual LAN will also allow more accurate measurements with regards to the training time of the ANN, the number of Uncertainties passed through the network, and the processing time of packets while filtering.

Once the LAN has been set up, and the ANN and firewall have been tested, another possible experiment could be run: To have the ANN eventually independently function in place of the firewall. The independent ANN's behavior to network traffic can be observed and will determine how a more efficient filtering mechanism can be implemented.

Another modification to the ANN that should be implemented is the ability to learn the ANY option while filtering packets. Implementing the ANY option will allow the ANN to be more accurate during filtering. This will enable the ANN to take into consideration that a single rule may apply to multiple input values. The ANY option is an important part of the firewall, allowing network administrators to create rules more easily for a single type of expected traffic. Implementing the ANY option with the ANN will prevent the misclassification of traffic entering the network.

With these modifications to the ANN, more security mechanisms can be implemented. One such mechanism is attack detection. The ANN may be modified to be able to detect attacks over a LAN, such as flooding attempts. Essentially, in this case, the ANN can be modified further to integrate functionalities similar to intrusion detection, alerting network administrators of possible attack attempts. In this case, much like what was discussed in [5], may act as an anomaly-based detection tool to report suspicious behavior. Detecting attacks may also be beneficial for public networks, since the results of the experiments show that the ANN can be used to filter packets in public networks with positive results. Cybercriminals sometimes use public networks to conceal themselves in the high levels of traffic. Attack detection along with the ANN may aid authorities in detecting these criminals.

Another security mechanism that can be implemented is a way to send any packets labeled as Uncertain back to the firewall for re-filtering. Allowing these Uncertainties into the network may pose a risk to the integrity of the network, opening opportunities for attacks to occur. Alternatively, benign traffic may also be blocked due to being labeled Uncertain. Implementing methods to handle Uncertainties will solve the issues that arise because of them, preserving the integrity of the network.

ANNs may also be compared to other ML frameworks. As seen in Section 3.1, decision trees and SVMs have also been experimented with for enhancing traffic filtering capabilities. In the future, these two methods could be designed and developed for the same purpose as mentioned in this thesis: To improve the firewall’s default rule policy for data packets that do not match any filter rules. Certain aspects of each method can then be analyzed and compared to determine the ideal framework for this application, such as processing time, memory usage, and classification accuracy.

Chapter 7: Conclusion

The work done in this thesis explores the idea of implementing feed-forward ANNs to work alongside firewalls in order to improve their filtering capabilities. With the available time and resources, a simulation was designed to determine if this is possible. To do so, tests were done with three options for the number of rules: 4, 9, and 15 rules. All sets of rules also included a default rule. The tests were completed with both default rule cases: ACCEPT or DENY.

In order to complete the experiments, the rules alone were used to generate training files for the ANN. This is the information that is known by the system with the exact actions that need to be taken against those packets. The testing examples were generated randomly, using the given network parameters as limits. The examples found in the testing file may be known from the firewall rules, or they may be unknown by the network. The results of the testing phase of the ANN were noted down to be compared with how a firewall might react to the testing file.

Given the results of the tests done with different cases, it was found that ANNs may indeed be helpful in improving filtering capabilities. The results show that ANNs are capable of classifying network packets almost as well as firewalls with a higher number of rules. This can be used to enhance the firewall’s filtering capabilities, especially by improving the default rule’s issue with wrongly filtering packets that do not match any firewall rule. With further modifications, ANNs can be used to refilter the packets that are subject to the default rule.

The results of this thesis pave the way for further improvements of this system and open the doors to the possibility of using AI in information and network security. These improvements will not only eventually perfect the packet filtering capabilities of firewalls but may also include security mechanisms that can prevent attacks from happening before they even enter the network. This can essentially combine firewalls and intrusion detection/prevention systems into a single system.

References

[1] SANS, "German Steel Mill Cyber Attack", 2014. [Online] Available: https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf

[2] K. Valentin and M. Maly, “NETWORK FIREWALL USING ARTIFICIAL NEURAL NETWORKS,” Computing and Informatics, vol. 32, pp. 1312–1327, 2013.

[3] A. Ng, "Machine Learning". [PowerPoint Presentation], Stanford University, 2017

[4] M. Jordan and T. Mitchell, "Machine learning: Trends, perspectives, and prospects", Science, vol. 349, no. 6245, pp. 255-260, 2015.

[5] E. Reddy, "Neural Networks for Intrusion Detection and Its Applications", Proceedings of the World Congress on Engineering, vol. 2, no. 5, 2013.

[6] M. Ussath, D. Jaeger, F. Cheng and C. Meinel, "Identifying Suspicious User Behavior with Neural Networks", 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), 2017.

[7] P. Wang, "Packet classification with multiple decision trees", 2015 21st Asia-Pacific Conference on Communications (APCC), 2015.

[8] M. Wahid and A. Abdullah, "Detecting an Anomaly Behavior through Enhancing the Mechanism of Packet Filtering", Journal of Computer Science, vol. 11, no. 6, pp. 784-793, 2015.

[9] A. Ng, Machine Learning: Supervised Learning [PowerPoint Presentation], Coursera: Standford University, 2017.

[10] C. McDonald, "Machine learning fundamentals (I): Cost functions and gradient descent", Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220. [Accessed: 01- Mar- 2018].

[11] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. New York: Cambridge University Press, 2014.

[12] K. Murphy, Machine learning: A Probabilistic Perspective. Cambridge, Mass. [u.a.]: MIT Press, 2014.

[13] I. Goodfellow, Y. Bengio and A. Courville, Deep learning. Cambridge (EE.UU.): MIT Press, 2016.

[14] C. McDonald, "Machine learning fundamentals (II): Neural networks", Towards Data Science, 2018. [Online]. Available:

https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef. [Accessed: 01- Mar- 2018].

[15] K. Brantley, "BCAP: An Artificial Neural Network Pruning Technique to Reduce Overfitting", Master Thesis, University of Maryland, Baltimore County, 2016.

[16] B. Anderson, Computational neuroscience and cognitive modelling. Los Angeles [u.a.]: SAGE, 2014, pp. 81 - 96.

[17] J. Schmidhuber, "Deep learning in neural networks: An overview", Neural Networks, vol. 61, pp. 85-117, 2015.

[18] D. Kriesel, A Brief Introduction to Neural Networks. 2007. [Online] Available: http://www.dkriesel.com/en/science/neural_networks

[19] R. Garreta, "Artificial Neural Networks & Backpropagation," [PowerPoint Presentation]. Academia, 2016.

[20] M. Cilimkovic, "Neural Networks and Back Propagation Algorithm", Institute of Technology Blanchardstown.

[21] F. van Veen, "The Neural Network Zoo - The Asimov Institute", The Asimov Institute, 2016. [Online]. Available: http://www.asimovinstitute.org/neural-network-zoo/. [Accessed: 02- Mar- 2018].

[22] "Unsupervised Feature Learning and Deep Learning Tutorial", Ufldl.stanford.edu. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Accessed: 02- Mar- 2018].

[23] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional Neural Networks for MATLAB", in MM '15 Proceedings of the 23rd ACM international conference on Multimedia, Brisbane, Australia, 2015, pp. 689 - 692.

[24] "FANN", FANN. [Online]. Available: http://leenissen.dk/fann/wp/. [Accessed: 02- Mar- 2018].

[25] A. Pavelka and A. Prochazka, "Algorithms for initialization of neural network weights," in Sbornk prspevku 12th rocnku konference MATLAB, vol. 2, pp. 453 – 459, 2004.