Eye of Sauron: Using Neural Network for Anomaly Network Intrusion Detection

Arizona State University CSE 548

Derbique, Austin
aderbiqu@asu.edu

Leyba, Kirtus
kleyba@asu.edu

Schmidt, Ryan
raschmi4@asu.edu

April 3, 2021

Abstract

Machine learning based approaches to detecting anomalous or malicious network traffic are necessary in today's age with sophisticated attackers capable of evading typical firewall rules or signature-based matching of exploits. We examine using a feedforward neural network and self-organizing map to classify network traffic as either benign or as traffic that constitutes an attack. We evaluate these neural networks by training them on a variety of test data and see how their performance differs under varying circumstances. This helps to provide a deeper understanding of the features that make for a successful machine learning based anomaly detection system when applied to real-world traffic.

Keywords— Neural Networks, Machine Learning, Anomaly Detection, Network Intrusion

I. Introduction

Today's networks are increasingly under attack. While stateful firewalls can control whether or not traffic gets blocked, they cannot determine whether any particular packet or connection is benign or malicious beyond examining the connection state combined with data from the current packet only [2]. As such, there is a need for utilities to examine these traffic flows to determine when a compromise might be happening. A typical approach to do this is to make use of an Intrusion Detection System (IDS). An IDS operates by examining the network traffic against a database of known exploit signatures, and raises alerts whenever traffic matches on of these signatures [4].

Signature databases fall short in detecting novel attacks, and modifications of attacks designed to evade the signatures. This is where anomaly detection comes into play. Because the malicious traffic often does not fit to the same patterns used for benign traffic, we can say it is anomalous. By using an anomaly detection system, we therefore are able to perform this classification of traffic that typical stateful firewalls and IDSes are unable to perform. Anomaly detection relies on Machine Learning (ML) to determine whether or not any particular network connection is malicious. In this paper, we will examine two ML approaches to anomaly detection and discover how well they perform.

Demonstration Video

The video presentation can be found on YouTube at https://youtu.be/bAF4Q7AWtIE.

II. System Models

A. *System Model*

We designed two neural networks, one Feedforward Neural Network (FNN) and one Self Organizing Map (SOM). An FNN consists of layers of neurons hooked together in a directed acyclic graph, so that data progresses from the input layer through the hidden layers, if any, and finally to the output layer. Due to the acyclic nature of the connections, information flows in only one direction. This is illustrated in Figure 1. To train an FNN, the technique we utilize is back propagation. Our training data is labelled with the correct responses, so by running the data through the FNN and comparing its output with the expected output, we can determine where the network needs adjustment. The back propagation algorithm takes these incorrect responses and calculates an error

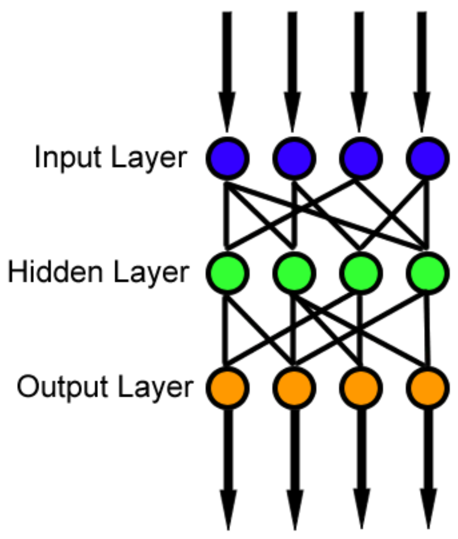


Figure 1: Diagram of a feedforward neural network. Data flows through the neurons in a single direction only. Image credit: Wikipedia [5].

function, feeding the calculated value backwards through the network to adjust the parameters of the layers, so that next time it will generate more correct responses for that data. Over multiple training cycles, the network will converge on neuron weights that minimize the error. A major risk of training FNNs is overfitting, in which the network learns the training data perfectly but loses any generalizations that allow it to accurately determine the categorization of unknown data.

Meanwhile, an SOM is typically used to reduce the dimensionality of a dataset. Commonly, it is used to create a 2-dimensional map of some higher-order data while retaining the general topology of the original data. During the training process, similar data is grouped together in a competitive process that does not require knowing the correct classifications of the data beforehand. This training mode is known as unsupervised training, because the neural network is creating its weights from training data that was not manually labelled with correct response beforehand. Instead, it extracts features based on the structure of the data. Once training is complete, new data is matched with the closest weight vector of the SOM to determine its classification.

B. Software

Various tools we have used are as follows:

- Python 3.8 is the language used to write our neural networks.
- The *tensorflow* and *keras* libraries for Python are used to provide the underpinnings of the neural networks.
- Additional Python libraries include *sklearn*, *pandas*, *numpy*, and *matplotlib* for parsing, preprocessing, and viewing the data.

C. Security Model

For an Intrusion Detection System to provide the most benefit, it must analyze traffic as it is flowing over the network. Our security model assumes that our neural networks are able to view this traffic stream live and be able to classify traffic as benign or malicious. If our neural networks discover malicious traffic, it would be up to some other system to determine what to do with it, such as blocking the traffic, sending alerts, or doing something more sophisticated such as various moving target defense schemes. In any event, what happens with the traffic after our neural network detects it is not part of our security model.

For our security model to work, we must be able to analyze all relevant traffic; if traffic bypasses our neural network then our classifications may be incorrect for the traffic that we do see, or we may never detect malicious traffic due to never being able to see it.

III. Project Description

Our project involves creating a handful of neural networks: FNN, various RNNs, and a SOM and testing them with two distinct datasets. This will let us know how these types of networks compare against each other and where the strength's of each network lie.

A. *Project Overview*

We originally planned to work on the first and third parts outlined by the project specification: Anomaly Detection Accuracy with Unknown Attacks and Unsupervised Training for Detecting Network Attacks. Each of these parts has multiple components and deliverables associated with them, which are further broken down below.

Due to completing these two parts ahead of schedule, we additionally worked on the second part outlined in the project specification: FNN vs RNN.

B. *Task 1: Set up Part 1 Framework*

This task involves setting up the framework for Anomaly Detection Accuracy with Unknown Attacks by creating the python file, downloading the training data, and building the framework to save and load trained models for comparison purposes. Finally, the framework structure will be put on GitLab.

C. *Task 2: Collect Part 1 Data*

With the framework complete, we will need to train the network on subsets of the training data, then run the testing data against the trained models to see how they react. We will do this for multiple subsets and determine for each its accuracy and how it compares against the other subsets.

This task depends on Task 1 to be completed first.

D. *Task 3: Complete Part 1 Deliverables*

With the part 1 data collected, we will need to write the results in the report, create relevant figures, and create the demo video.

This task depends on Tasks 1 and 2 to be completed first.

E. *Task 4: Set up Part 2 Framework*

This task involves building the FNN and a handful of RNN models (simple RNN, GRU, LSTM) and training/testing them to determine their relative performance on two different datasets. One dataset has time series data and the other does not.

F. *Task 5: Collect Part 2 Data*

By running each of the neural networks and determining their relative performance, we can collect data related to this task. Additionally we will vary the lookback on the RNNs from 1 to 5 to determine how that parameter impacts the accuracy and loss of the networks.

G. *Task 6: Complete Part 2 Deliverables*

With the part 2 data collected, we will need to write the results in the report, create relevant figures, and create the demo video.

This task depends on Tasks 4 and 5 to be completed first.

H. *Task 7: Set up Part 3 Framework*

This task involves setting up the framework for Unsupervised Training for Detecting Network Attacks by creating the python file and downloading the training data, then putting the framework structure on GitLab.

I. *Task 8: Collect Part 3 Data*

With the framework complete, we will need to train the network. This will be done using the Day 5 data, with 50% of the data used for training and the other 50% used for testing to determine the network's ability to detect port scans. With the test data, we will determine the true positive, false positive, true negative, and false negative rates of the network.

This task depends on task 7 to be completed first.

J. *Task 9: Complete Part 3 Deliverables*

With the data collected, we need to format it for inclusion in the discussion section, including creating relevant figures or tables. Additionally, research will need to be made into the pros and cons of using unsupervised training for network attack detection. Finally, we need to create the demo video.

With exception of the research component, this task depends on tasks 7 and 8 to be completed first.

K. *Task 10: Complete Midterm Report*

At the time of the midterm, we need to evaluate how far along we are and how that compares to our projected timeline, then provide relevant discussion on our current progress and things we have learned.

L. *Task 11: Complete Final Report*

For the final report, we need to complete all deliverables and include them in the report. Additionally, we need to flesh out the shared sections, updating the introduction, system model, discussion, and conclusion with what we have learned.

This task depends on tasks 3, 6, and 9 to be completed first.

M. *Task 12: Edit and Upload Demo Videos*

After recording both demo videos, we need to edit them to ensure they remain within the alloted time limit of 10 minutes each and to make the videos flow better. Once edited, they will each be combined and then uploaded to YouTube as a single video so that other class members can do their peer evaluations.

This task depends on tasks 3, 6, and 9 to be completed first.

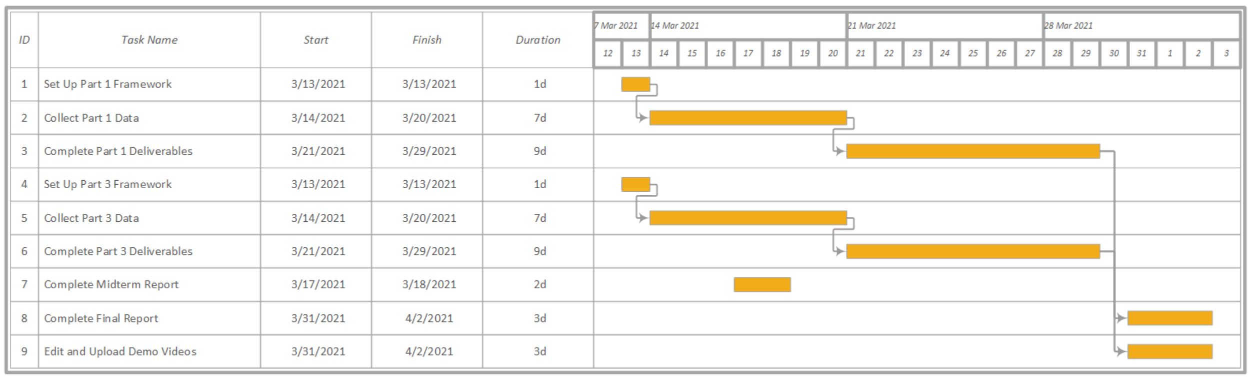
N. *Project Task Allocation*

Our team will work together on all tasks, however, we will allocate the following tasks to task leaders to lead the effort and find solutions to obstacles in the way of getting the task completed. The overall workload breakdown for each task is given in Table 1. We have elected **Austin Derbique** as the project lead.

Task	Task Leader
Set up Part 1 Framework	Austin Derbique
Collect Part 1 Data	Kirtus Leyba
Complete Part 1 Deliverables	Ryan Schmidt
Set up Part 2 Framework	Kirtus Leyba
Collect Part 2 Data	Ryan Schmidt
Complete Part 2 Deliverables	Austin Derbique
Set up Part 3 Framework	Kirtus Leyba
Collect Part 3 Data	Ryan Schmidt
Complete Part 3 Deliverables	Austin Derbique
Complete Midterm Report	Kirtus Leyba
Complete Final Report	Austin Derbique
Edit and Upload Demo Videos	Ryan Schmidt

Table 1: The workload distribution of the project

O. *Project Timeline*



ID	Task Name	Start	Finish	Duration
1	Set Up Part 1 Framework	3/13/2021	3/13/2021	1d
2	Collect Part 1 Data	3/14/2021	3/20/2021	7d
3	Complete Part 1 Deliverables	3/21/2021	3/29/2021	9d
4	Set Up Part 3 Framework	3/13/2021	3/13/2021	1d
5	Collect Part 3 Data	3/14/2021	3/20/2021	7d
6	Complete Part 3 Deliverables	3/21/2021	3/29/2021	9d
7	Complete Midterm Report	3/17/2021	3/18/2021	2d
8	Complete Final Report	3/31/2021	4/2/2021	3d
9	Edit and Upload Demo Videos	3/31/2021	4/2/2021	3d

Figure 2: Gantt chart of original project timeline; we completed all tasks ahead of schedule and were able to finish Part 2 as well.

IV. Discussion of Anomaly Detection Accuracy with Unknown Attacks

Trained On	DoS Accuracy	Probe Accuracy	U2R Accuracy	R2L Accuracy
DoS	0.999	0.371	0.000	0.009
Probe	0.718	0.988	0.000	0.001
U2R	0.000	0.000	0.000	0.000
R2L	0.691	0.133	0.115	0.768
DoS, Probe	0.981	0.986	0.019	0.015
DoS, U2R	0.998	0.355	0.538	0.033
DoS, R2L	0.977	0.375	0.019	0.279
Probe, U2R	0.787	0.987	0.288	0.006
Probe, R2L	0.854	0.976	0.038	0.560
U2R, R2L	0.700	0.076	0.692	0.862

Table 2: This table shows the results of training the FNN model on different subsets of attacks (Shown in the Trained On column). The rest of the columns show how the model performed when tested on different attack categories.

How does the FNN model handle new attacks? We investigated the accuracy of the trained FNN model on new attacks by breaking the NSL KDD dataset into separate subsets for testing and training. An overview of these results is shown in table 2. The model always performs best on attacks it is trained to detect, which matches expectations. Interestingly, some new attack subsets of the data are detectible. For instance, we see that if we train a model to detect probe attacks, it is reasonable at also detecting DoS attacks (with an accuracy of 0.718).

What is the model's behavior in terms of accuracy for new attacks? We can further interpret these results by noting that some features in the data allow the model to infer new attacks in some cases even without training on those attacks of that specific type.

How does the change in attack types for training impact the model? When we trained on larger subsets, for instance (Probe, U2R), the model did better on the untrained attack categories than with a single training subset. This indicates that the model can learn some common attack patterns with more information rich data.

How do new attacks differ from trained attacks? To investigate what makes the attacks novel to the model we investigated the dataset. We performed principal component analysis (PCA) on the dataset as shown in figure 3 to visualize the differences between attack subsets. Notice that DoS and Probe seem to have a lot of overlap, aside from a special case of DoS attacks seen in the bottom right of the figure. Additionally, there is a large amount of overlap between U2R and R2L attacks. This reflects our other results in the sense that it appears training the FNN model on similar attacks allows the model to detect novel attacks that are similar.

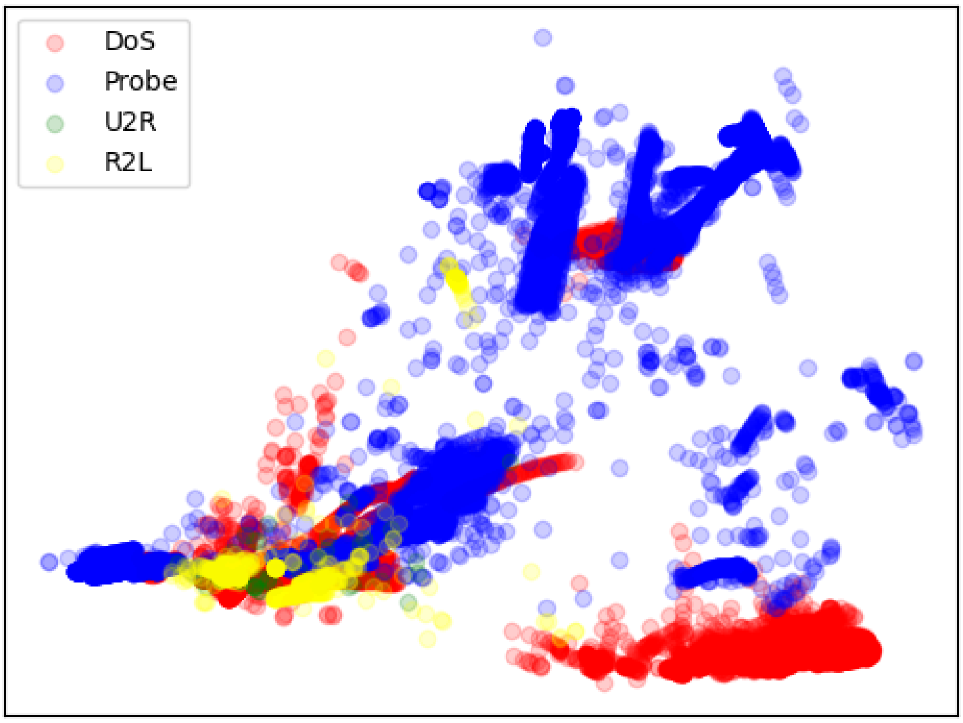


Figure 3: Principal Component Analysis (PCA) on the NSL KDD Train+ dataset. We use PCA to reduce the data dimensionality to visually compare the four main attack categories.

Does prediction accuracy relate to the attacks being similar? We found that if two attack subsets were similar then if we trained a model with one, we would be more effective at detecting the other. This is hugely important for the NSL KDD dataset because the U2R subset is much smaller than the other subsets. Note that the model trained purely on the U2R attack subset was by far the worst performing, but the model trained on both U2R and R2L attacks could actually detect U2R attacks with much higher accuracy. The richness in similar attack subsets can make a more robust intrusion-detection model!

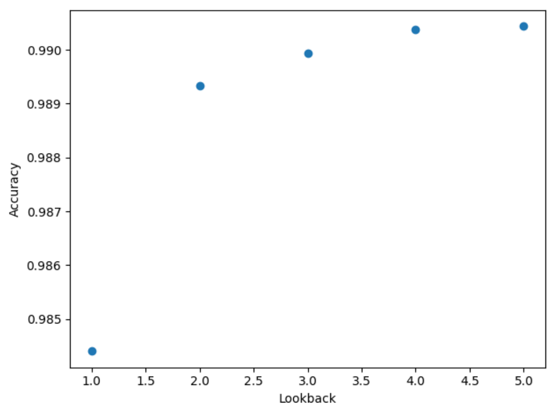
V. Discussion of FNN vs RNN

For task two we tested more advanced neural network topologies on the task of network attack detection. More specifically, we experimented on the impact of recurrent neural networks on time-series data and evaluated their performance against traditional neural networks. We tested four types of neural networks for this task: a simple ANN, a simple RNN that naively loops over previous samples, a gated recurrent unit (GRU) [1], and a long short term memory (LSTM) neural network [3].

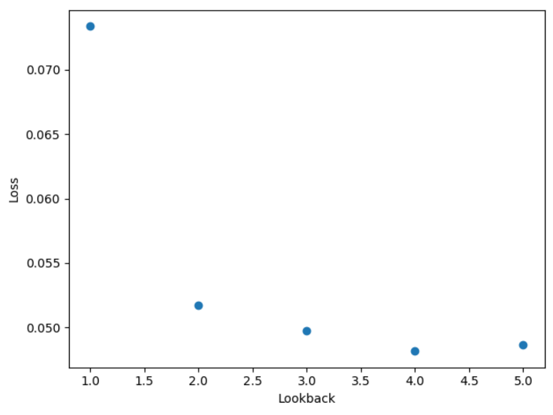
Are RNNs better than ANNs in detecting the time-series data such as network flows? We found that RNNs consistently outperformed ANNs on time-series based data. We used the IDS DDoS and IDS PortScan datasets as examples of time-series data. Each of these datasets represent PCAP traffic files that show traffic flows and other time related data. As shown in table 3, we found that GRU networks performed the best on the IDS DDoS dataset and that LSTM networks performed the best on IDS PortScan networks.

Model	Dataset	Accuracy	Loss
SimpleRNN	KDD	0.5465	0.6922
LSTM	KDD	0.5607	0.6894
GRU	KDD	0.5582	0.6892
ANN	**KDD**	**0.5714**	**0.6836**
SimpleRNN	IDS DDoS	0.9988	0.0073
LSTM	IDS DDoS	0.9983	0.0693
GRU	**IDS DDoS**	**0.9989**	**0.0076**
ANN	IDS DDoS	0.9976	0.0881
SimpleRNN	IDS PortScan	0.9885	0.0529
LSTM	**IDS PortScan**	**0.9905**	**0.0494**
GRU	IDS PortScan	0.9904	0.0479
ANN	IDS PortScan	0.9822	0.0818

Table 3: Performance of various neural network models on multiple datasets. The best result for each dataset is in bold.



(a) Accuracy as a function of lookback.



(b) Loss as a function of lookback

Figure 4: The impact of lookback on recurrent neural network performance. Figure 4a shows the final loss for testing an LSTM as lookback increases from 1 to 5, and figure 4b shows the loss as the lookback increases.

What types of RNNs have capabilities to identify long-range dependencies and how they achieve it? RNNs in general lookback a certain distance in the input to the network and use previous remembered samples to impact the current classification. Simple RNNs struggle with longer time periods between related data, but this is mitigated by the various memory and forget gates in the LSTM neural network model [3]. A network model that doesn't use as many gates but still shows high effectiveness at interpretting distant temporal relationships is the GRU [1].

In our experimental studies, we found that increasing the lookback of the recurrent neural networks increased overall testing accuracy. This is demonstrated in figure 4. Note that as we increased the lookback the accuracy for the network increased and the loss decreased. We found that there were diminishing returns eventually, which indicates there is a limit to how distant temporally relevant samples are to attacks.

For one type of RNN identified in (b), what observations have been made with increased long-range dependency mapping concerning the accuracy of the chosen RNN. It has been shown that LSTMs outperform many other neural network types when it comes to longer term temporal relationships in time series data [3]. This means that long-range dependent samples in the data can still impact classification long after a related sample is seen. This is what motivates the use of LSTMs on long temporal data such as language processing and intrusion detection systems.

Were RNNs found to be better (or NOT better) than ANNs? Does it have anything to do with the dataset used? We found that RNNs outperformed ANNs specifically on time-series data. As shown in table 3, the GRU based network did the best on the IDS DDoS time series data and the LSTM did the best on the IDS PortScan data. In the non time-series data however the ANN was competitive and outperformed all the RNNs for this experiment.

VI. Discussion of Unsupervised Training for Detecting Network Attacks

In task three, we implement a self organizing map (SOM) to perform unsupervised learning on the NSL dataset. A SOM is an artificial neural network that is trained in an unsupervised mode. The purpose of a SOM is to group clusters of similar nodes of high dimensions such that the resulting graph's dimensions are reduced in order to make better sense of the data.

What are the pros and cons of using an unsupervised training mode for network attack detection? One of the main benefits to unsupervised training is the ability to abstract the model from specific types of attacks and look at the larger picture. This generalizes the attack detection technique allowing for the methodology to be used on a broader range of network attack techniques. Another benefit is that unsupervised learning is great for attacks that are not perfectly labeled. Because the ANN is not trained on label data, the network is capable of making decisions with higher degrees of accuracy for outlying data than a model trained using labeled data. One of the largest drawbacks of unsupervised learning is the difficulty with accuracy. On occasion, training time can also take longer as data is not clearly labeled, therefore requiring the ANN to perform more computations in order to generate a result.

Observe the performance and accuracy of SOMs while attempting to identify port scans from normal user activity. Our tests concluded that the performance and accuracy of our self organizing map was proportional to amount of neurons used on the grid. Creating a larger grid with more neurons led to significantly longer processing time with only minor improvements in accuracy. Reducing the size of the grid took less time to process, but lowered the accuracy rate. In Figure 5 shows results for a grid with size 44x44 rows and columns represented in two dimensional space. This size appears to be the best balance between performance and accuracy. In the graph, The blue dots represent benign data and the red dots represent the port scans. Overall, the SOM performed quite well in classifying benign data from port scans. One observation are the purple dots where the data was labeled both as benign and port scan. This is likely due to the model being trained in an unsupervised mode, where there are no labels used to dictate a boolean state of attack.

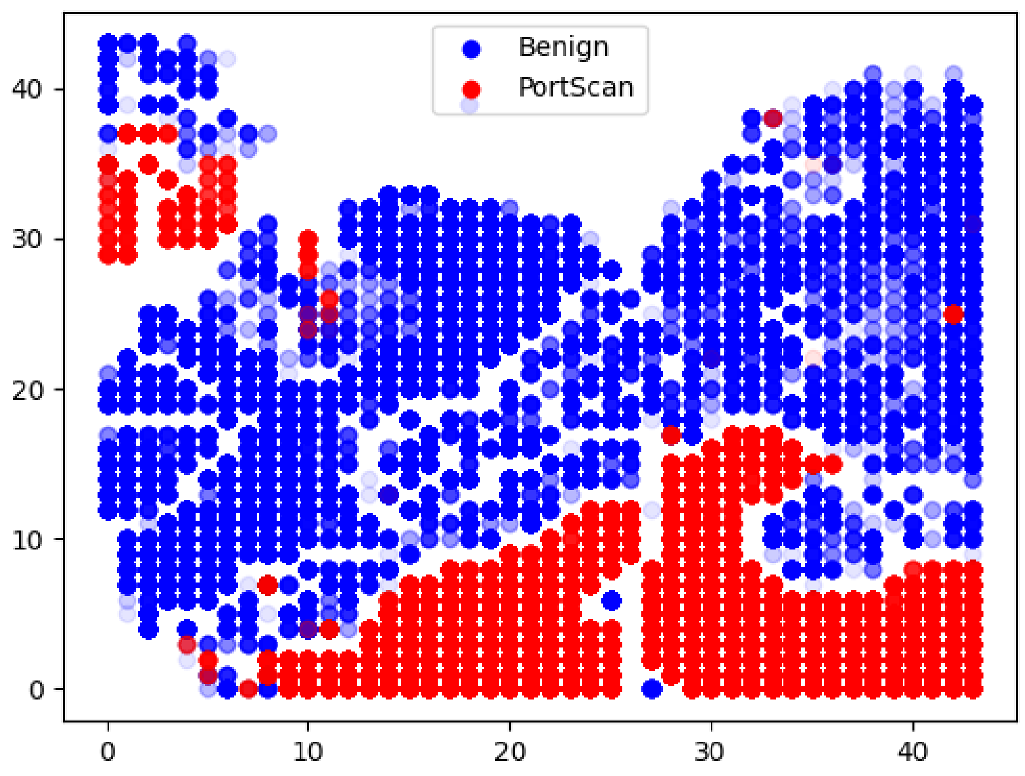


Figure 5: Traffic Classification using Self Organizing Map (SOM) on the NSL KDD Train+ dataset to predict whether traffic is benign or a port scan.

Are SOMs a good choice for network attack detection? Explain and Why and Why Not? Our results conclude that overall, SOMs are a great choice for network attack detection. Though, it is difficult to know from this test how well a SOM would perform in identifying specific attacks from other attacks. This would require more evaluation to conclude, but for at least detecting an attack in general, the model performs quite well.

What are the true positive and false positive rates observed using SOM for network attack detection? As seen in figure 6, blue dots represent True Positives/True Negatives (TP/TN) and are considered correct, while the orange dots represent False Positive/False Negatives (FP/FN) which are incorrect. This is measured by stripping the dataset of labels, ingesting the data into our SOM, and then comparing the predicted results to the original dataset with labels. Results show that the SOM performs well at identifying results accurately for network attack detection.

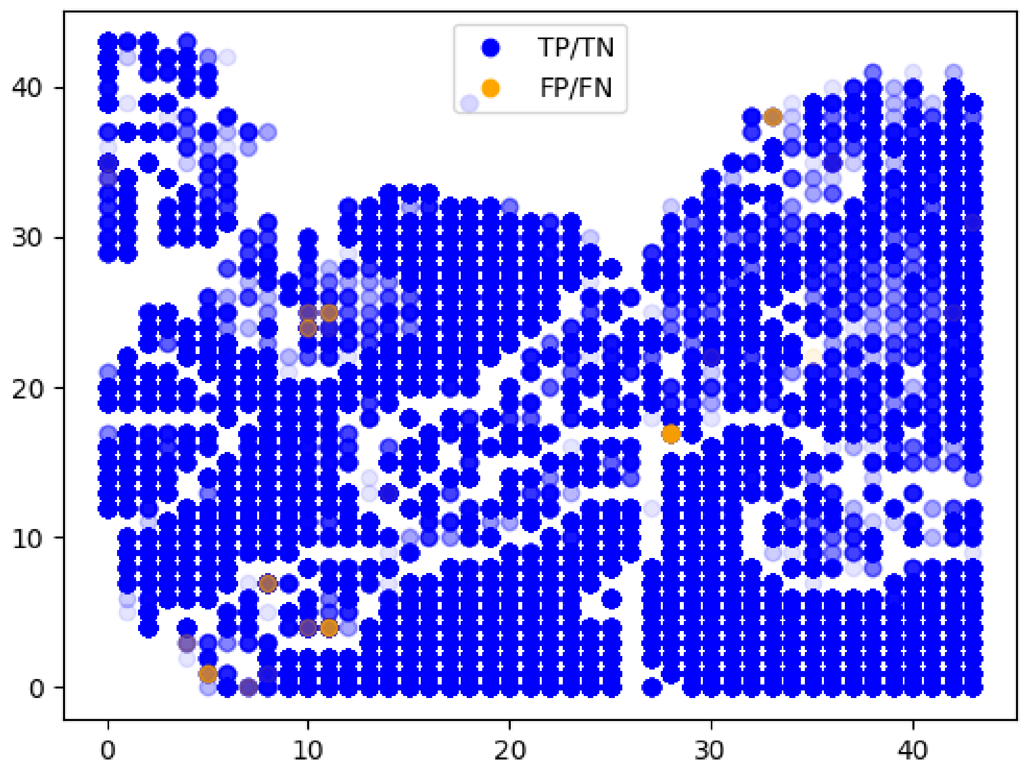


Figure 6: Accuracy of True Positive/True Negative (TP/TN) versus False Positive/False Negative (FP/FN) results using Self Organizng Map (SOM) on the NSL KDD Train+ dataset.

VII. Conclusion

In this project, we implemented various machine learning approaches for detecting anomalous or malicious network traffic. By using the provided KDD datasets, we evaluated performance and accuracy of simple ANNs, simple FNNs, LSTMs, GRUs, and SOMs. In task one, we discussed anomaly detection accuracy with unknown attacks, breaking down accuracy by type of attack combination. In the second task, we discussed feed forward networks from recurring neural networks, and various pros and cons of each. We observed what happens when lookback time is increased as well as what happens when switching time series data for non-time series training data. Finally, we implemented and evaluated unsupervised training on self organizing maps (SOMs) to study performance and accuracy for unlabeled datasets. In conclusion, we implemented and evaluated a broad range of different machine learning approaches and successfully identify the strengths and weaknesses of each.

References

[1] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[2] Fortinet. *Stateful Firewall.* 2021. URL: `https://www.fortinet.com/resources/cyberglossary/stateful-firewall` (visited on 03/19/2021).

[3] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to forget: Continual prediction with LSTM”. In: (1999).

[4] Palo Alto Networks. *What is an Intrusion Detection System?* 2021. URL: `https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids` (visited on 03/19/2021).

[5] Paskari. *Feed forward neural net.gif.* Nov. 2006. URL: `https://en.wikipedia.org/wiki/File:Feed_forward_neural_net.gif` (visited on 03/19/2021).