SAYEDAHMED, SAED, M.S. Predicting Network Anomalies with Deep Sequence Analysis. (2019)
Directed by Dr. Somya D. Mohanty. 52 pp.

Network attacks can be very costly to victims and due to the complexities in they disparate types of attacks they are also hard to detect/predict. Understanding the underlying network traffic is critical in the developing automated solutions which can prevent such attacks in future. Within our study, we develop data-driven machine learning approaches to detect and predict such attacks based on the traffic behavior. Our study compares the differences in detection versus prediction of attacks/network anomalies where we compare traditional machine learning models for detection to the developed approach of leveraging network traffic as sequences of states in order to predict future network behavior. We also provide a comprehensive comparison of the different approaches taken with a wide range of feature-sets, hyperparameters, and variables evaluated for detection and prediction accuracy.

PREDICTING NETWORK ANOMALIES WITH DEEP SEQUENCE ANALYSIS

by

Saed SayedAhmed

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2019

Approved by

Committee Chair

APPROVAL PAGE

This thesis written by Saed SayedAhmed has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair	______________________
	Somya D. Mohanty
Committee Members	______________________
	Jing Deng
	______________________
	Stephen Tate

____
Date of Acceptance by Committee

____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to thank Dr. Somya D. Mohanty for supervising this thesis. His guidance and support have been very helpful, and it allowed me to learn a lot, improve my skills, and add up to my professional experience. I would like to thank Dr. Jing Deng and Dr. Stephen Tate for taking the time to read the thesis and be a part of the committee.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Malicious network behavior can be costly. It can lead up to a degradation in service availability, losing access to sensitive information, and may compromise privacy. With the rise of cloud services, the Internet of Things (IoT), and electronic government, it is essential to identify malicious behavior in a network environment to prevent such damages that can affect a wide range of users.

Machine learning techniques can help us develop systems that try to model malicious network behavior and detect it. It has already been utilized to solve such a problem [LWLS06] [ZZH08]. It can understand patterns and infer relationships better than the old rule-based techniques. Also, they are lightweight approaches that can keep learning new kinds of traffic actively. This makes them easily deployable and future-proof in many environments.

A common problem with traditional machine learning models that they cannot deal with sequential data representation. Sometimes this prevents the model from understanding the full contextual picture and may lead it to miss some information. Also, this limits them to being reactive approaches for attack detection.

Traffic data can be represented as a sequence of packets, connections, or time windows. A packet is the atomic representation of traffic data. A single connection is a group of a sequence of packets. While time windows can be an aggregation of either one over a specified period, therefore, the captured traffic can be modeled as a sequence of one of those representations. This can potentially help in not only in detecting and modeling malicious behavior in a network, but also in predicting it.

However, intuitively, traffic data over time can be thought of as a sequence of states that the network undergoes. Understanding these state changes which occur during normal operation of a network versus an attack/anomalous scenarios can provide essential information about predicting such events in the future. In comparison, traditional machine learning techniques are not suited for sequential data, as they interpret each input sample independent from previous ones. Within our approach we build and develop prediction architectures based on recurrent neural networks which leverage the sequential nature of states in network traffic. Such architectures can be trained to keep only the necessary state change information in order to predict the future events in a network. This makes them a good alternative for traditional machine learning techniques.

I.1. Contribution

The main contribution of this work is the study of different techniques when using machine learning and recurrent neural networks to detect and predict malicious traffic in network environments. In particular, we focus on those aspects:

1. We study the effect of different traffic representations on model performance. The study evaluates a wide range of network features and different aggregations strategies for feature engineering in network data.

2. We study the performance of traditional machine learning techniques in detecting malicious traffic behavior. In addition, we interpret the best models and explain them in order to feed the prediction methods.

3. We develop sequence modeling architectures based on neural networks to predict malicious behavior in the various network attack scenarios.

I.2. Thesis Content

This thesis is organized as follows: Chapter 2 introduces the necessary background for machine learning and deep learning. Chapter 3 provides the related literature to this research and explains some drawbacks in them. Chapter 4 explains the methodology and the utilization of the different techniques in solving the problem. Chapter 5 presents the results. Finally, Chapter 6, concludes and proposes future work and potential improvements.

CHAPTER II

RELATED WORK

In this chapter, we provide relevant literature for this thesis. We provide related work using machine learning to model network attacks and the related work to model traffic with recurrent neural networks.

Machine Learning has been widely used to detect or predict attacks in network environments [LWLS06] [SP14] [STG$^+$11]. The common approach is to do a binary classification (normal vs. attack) for detection using different machine learning approaches. For instance, in [LWLS06], they use probabilistic approaches, such as Bayesian Networks and Naive Bayes classifiers, to detect attacks and identify malicious hosts. They also use a decision tree to address the same problem. Also, [SP14] goes a little bit deeper and uses Logistic Regression and Support Vector Machines (SVMs). In their approach, they group traffic either by fixing the window length or the number of packets in a window. However, they stick to detecting attacks but not predicting them, and they do not apply sequence modeling techniques. Moreover, in [ZZH08], they use Random Forests to model traffic behavior patterns and indicates an ongoing attack. This allowed them to catch malicious behavior and explain why it happens.

In the last several years, neural networks became easier to implement and train due to the hardware and software rise. Graphics Processing Units (GPUs) became much powerful and ideal for neural networks training. While easy-to-use and regularly-maintained libraries, packages, and Application Programming Interfaces (APIs) made it even more straightforward to do the job. As a result, more research started adopting feedforward neural networks, RNNs, LSTMs, and GRUs. In [AS18] and [XSDZ18], they utilize recurrent neural networks (RNNs), Long-Short Term Memroy (LSTMs), and Gated Recurrent Units (GRUs) for sequence modeling. However, they only try to detect the attacks and malicious traffic but not predict it in advance. Furthermore, [KKTK16] utilize LSTMs to model sequences, but use one fixed sequence length in their experiments.

In [DAF18], they use machine learning and a feedforward neural network to detect and predict attacks. However, there is no sequence modeling done and the window length is fixed as well. Also, they do not provide a reasoning behind the limited number of features they engineer. This can potentially miss some information or make the model focus on using some features that are not as important as what can be obtained.

In [RRD18], they use LSTMs to predict several kinds of attacks. Though, they tried one sequence length and did not vary it across their experiments. They also predicted one step ahead but not more than that. Their experiments did not include neural networks with RNNs or GRUs, despite LSTMs not performing good in all the scenarios, especially with DoS attacks. Finally, to evaluate their baseline and LSTM models, they used one evaluation metric, which can be misleading sometimes.

In general, most of the approaches achieved excellent performance. However, those lack trying different window lengths or different sequence lengths. Also, all of them tend to do a binary classification between normal and attack traffic. Besides, a few tried to predict the attack, and to the best of our knowledge, none tried to predict more than one step ahead.

CHAPTER III
BACKGROUND

III.1. Dataset

In our analysis, we utilize the CTU University Botnet Dataset CTU-13 [GGSZ14]. The dataset includes real-time data captured within the university network, which includes normal, background, and malicious traffic. Normal traffic consists of packets from trusted sources, whereas botnet-driven attacks cause malicious traffic. The background traffic is unknown connections/packets which were not annotated. The authors developed a virtualized network mapped onto Microsoft Windows XP systems, where 13 different types of botnet attacks were conducted across different time intervals. Each virtual machine was also bridged to the university network in order to create a cohesive dataset that included real-world normal/background traffic.

Each attack is executed on a separate network application protocols. In this research, we choose to study the Distributed Denial of Service (DDoS) attack on Internet Relay Chat (IRC) protocol, and spam attacks over the same protocol. DDoS is an attack that runs from multiple client/bots which are actively trying to flood a server with bogus requests in order to overwhelm the computation on it. The goal of this flooding mechanism is to push the server to deny establishing connections or allocating resources to clients. Spam attacks are messages containing malicious links that can lead to a similar result depending on its content. On the other hand, IRC protocol is an application layer protocol used for text-based communication.

For each scenario, the traffic data is stored in two formats, Pcap, and NetFlow. Pcap is the standard network packet capture format. Each observation in this format represents a raw network packet. On the other hand, NetFlow is a widely-used standard for network traffic flow statistics. In this format, an observation corresponds to a connection between two entities in the network, which conveys aggregated statistics over multiple packets.

III.1.1. Pcap Format

As the standard packet capture format, Pcap captures packets in their raw IP packet format. The raw format allows for a full capture of network traffic which includes the header and payload. In the case of CTU-13, to protect the privacy of users, only the Ethernet header information was captured, which included TCP, UDP, or an ICMP header. The Ethernet header includes a source MAC address, a destination MAC address, and an Ethernet type field. The MAC addresses represent the physical addresses of the endpoints, and the type represents the protocol used next, which is IPv4 in all of this dataset.

An IPv4 header consists of multiple fields, which include — a total length field, a flags field, a fragment offset, a time-to-live (TTL) field, a protocol field, a header checksum field, and endpoints IP addresses. The total length field holds the total length value of the header and data. The flags field determines and controls fragmentation. The fragment offset is measured in 8-byte blocks to represent the offset from the first packet in the sequence of fragments. The TTL field determines the lifetime of the packet in a unit of hop count (the number of routers it can go through). The protocol field represents the protocol used next within the packet. In this dataset, it can only be TCP, UDP, or ICMP. The header checksum is used to check the correctness of the IPv4 header, as it may have been changed in transmission. Finally, the IPv4 source and destination addresses are represented in two independent fields to identify the sender and the receiver of a packet. Also, every packet is timecoded.

III.1.2. NetFlow Format

The NetFlow is a network protocol (developed by Cisco) used to monitor network traffic and collect information about it. It is less comprehensive than Pcap and includes only aggregated information of the connection-level between two entities.

Within this format, a record/observation represents a timecoded connection between two endpoints. Each connection has — duration field, a protocol field, endpoints IP addresses, endpoints port numbers, a direction of transmission field, endpoints type-of-service (TOS), a total packets field, a total bytes field, and a source byte field. The duration field represents the number of milliseconds the connection lasted. The protocol, endpoint address, and port number fields are the same as the aforementioned in Pcap format. TOS field specifies the quality of the connection, which includes multiple factors such as speed, precedence, reliability, and throughput. The total-packets field shows the number of sent and received packets between the two endpoints. The total bytes field represents the total size of the sent and received packets. Lastly, the total-source bytes field contains the number of bytes sent from the source address.

III.2. Machine Learning Algorithms And Models

Supervised machine learning is a machine learning branch that tries to find a mapping function f that maps a particular input (observations/features) X to its corresponding output (labels) Y using previously-seen examples (equation III.1). A supervised learning algorithm takes in labeled input and tries to compute the closest mapping function with minimal error. The goal is to use the mapping function to predict Y values for previously unseen data. Supervised learning problems can be divided into two categories; classification problems and regression problems. In classification, the outputs are discrete and belong to a countable set of classes, e.g., sick or healthy. In regression, the outputs are continuous real values that may not necessarily be limited, e.g., temperature values. This study falls under the classification problems category.

$$X \xrightarrow{f(X)} Y \tag{III.1}$$

In this section, we explain the different supervised machine learning algorithms used in our study. We start by explaining the decision tree that lays the basis for Random Forests and Gradient Boosting. Then, we explain how Logistic Regression works. Afterward, we explain how neural networks, recurrent neural networks, Long-Short Term Memory, and Gated Recurrent Units.

III.2.1. Decision Tree

A decision tree is a basic machine learning technique where a binary-tree hierarchy of questions leads up to different decisions [Bre17]. Each internal node in the tree contains a decisive question about a feature (e.g., temperature > 10). On the other hand, each leaf node in that tree holds a decision determining the corresponding label. Decision trees construct their internal and leaf nodes based on the distribution of values in each feature in correspondence to the labels. Figure III.1 shows an example of a decision tree that determines if one can go out (yes) or stay at home (no).

Furthermore, decision trees are non-parametric machine learning models, where the model assumes nothing about the data or the contained errors. Along with the hierarchy property, decision trees help identify linear, polynomial, or even the most stochastic patterns that exist.

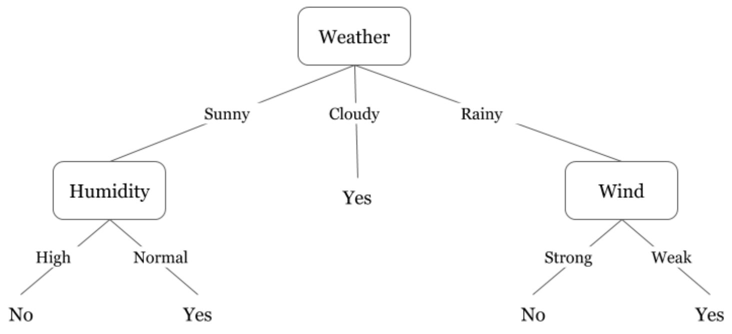


Figure III.1. A Decision Tree that Decides if you Can Go Out.

A decision tree training algorithm starts by looking at each feature against the class labels. It tries to find the best cut-off (question) that gives the purest split. A pure split divides different classes into the distinct branches of the node while keeping similar ones in the same branch as much as possible. For instance, for each features X_i, it looks for a x_i such that a question $X_i < x_i$ has the highest purity. Then, it compares the purity of the top question from each feature and chooses the highest one. It keeps recursively dividing each resulting node until it is highest purity is reached.

Decision trees use several split quality and purity measures. The most used ones are the Gini index and information gain. Gini index, shown in equation III.2, gives a direct indication of node's impurity. For each node, we add the squared probabilities of classes in a child node, then subtract that addition from 1. Then we can calculate the weighted average of all child nodes to get the final Gini value.

$$Gini = 1 - \sum_{i=C_0}^{C_N} P_i^2 \tag{III.2}$$

Information gain, as the name suggests, shows how much information can be given using an individual feature split. It does that by calculating the difference in entropy (Equation III.3) between the children and the parent nodes. Better splits have higher information gain.

$$Entropy = \sum_{i=C_0}^{C_N} -P_i \log P_i \tag{III.3}$$

Decision trees implement several tunable parameters that also affects training and branching. For instance, a node in a tree can stop branching if it has fewer observations than the specified minimum. In addition, a maximum number of leaf nodes can be set to deal with the same problem, and a maximum depth of the tree can also be specified. It can even go further and specify a minimum improvement in Gini or information gain to branch a node. Those parameters help the tree not only finish its construction faster but also counter overfitting a training dataset.

III.2.2. Logistic Regression

Logistic Regression, despite the name, is a statistical machine learning approach for classification. It fits an S-shape logistic function to the data and assumes a linear relationship between the features and the labels. Also, it assigns a single weight for each feature and global intercept for the whole fit.

Logistic Regression transforms the data from a probabilities plane to a log-of-odds plane. A probabilities plane plots the features against plain labels (0 or 1 in binary-classification). Then, it takes the log-of-odds (equation III.4) for those values to transform them into a log-of-odds plane.

$$\log_{odds}(y) = \log(\frac{y}{1-y}) \tag{III.4}$$

The zero becomes $-\infty$, and the one becomes ∞. Afterward, Logistic Regression creates a candidate line to fit the data. Although the points are located at $-\infty$ and ∞, we project them on this candidate line. Then, we convert back to the probabilities plane using equation III.5. This gives us the S-shape function. Subsequently, we calculate the log of maximum likelihood estimate (MLE) of the points on the S-shape function using equation III.6. The higher the value, the better the performance.

$$y\prime = \frac{e^{log_{odds}(y)}}{1 - e^{log_{odds}(y)}} \tag{III.5}$$

$$\log(MLE) = \sum_{i=0}^{N_y} \log(y_i - y_i\prime) \tag{III.6}$$

After calculating the log of MLE, we rotate the line in the log-of-odds plane and calculate the log of MLE again. From this rotation, Logistic Regression can infer in which direction it should rotate the line to improve the log of MLE. After several rotations, Logistic Regression can find the best fit, and it finishes its training. The S-shape function, which corresponds to the best fit, represents the Logistic Regression model.

III.2.3. Gradient Boosting

Gradient Boosting is an ensemble machine learning technique [Fri01]. An ensemble technique combines multiple weak learners in order to get a stronger model. In this case, Gradient Boosting trains multiple decision trees in a boosting manner, and boosting means that it directs each tree to learn from the mistakes made by its predecessor. At decision and prediction phase, each tree gives a decision, and a weighted vote occurs between the trees to give a final decision.

Initially, we start by a leaf-tree. The tree has a single log-of-odds value for all entries. It is calculated using the log-of-odds of the positive class by equation III.4. Afterward, we calculate the corresponding probability value using equation III.5. With this probability value, we calculate the residual value for each entry in the dataset using equation III.7.

$$r_i = y_i - y\prime_i \tag{III.7}$$

After we get the residual values, we build another decision tree to predict those residual values. However, the new log-of-odd values in the a new tree are calculated using equation III.8. Those log-of-odd values are calculated per leaf, with r_i representing a residual value in the leaf, and $y\prime_i$ representing the corresponding probability value from the tree.

$$log_{odds}(leaf) = \frac{\sum r_i}{\sum y\prime_i \cdot (1 - y\prime_i)} \tag{III.8}$$

Now that we got a new tree, we calculate the cumulative log-of-odds values per entry. This is calculated using equation III.9, with α representing the learning rate. The learning rate weighs what we learn from a new tree. To obtain the probability for this new log-of-odds value, we use equation III.5.

$$log_{odds}(y) = log_{odds}(y)\prime + \alpha \cdot log_{odds}(leaf) \tag{III.9}$$

In each new tree, we follow the same process of calculating residual values, log-of-odds values, and probability values. We keep creating new trees until there is no improvement noticed, or the maximum number of trees is fulfilled. After we are done creating trees, the final model consists of all the trees combined. Any new prediction, to be made, would go through all the trees and would take the summation of all their output values scaled by the learning rate, plus the output value from the initial leaf-tree.

As we saw, Gradient Boosting contains multiple hyperparameters that we can optimize. The learning rate (α) and the maximum number of trees are the most critical ones. Also, we can change our loss function from being the residuals. Other decision tree hyperparameters, such as the maximum depth of a tree and the maximum number of leaves in a tree, can be tweaked.

III.2.4. Random Forest

Random Forest is another ensemble machine learning technique [Bre01]. In contrast to Gradient Boosting, a Random Forest trains multiple decision trees independent of each other and performs a majority-vote for prediction. This allows for faster training compared to Gradient Boosting, as we can train multiple trees at once.

At first, a Random Forest algorithm creates subsampled datasets equal to the number of trees to be trained. Each subsample contains entries from the original dataset randomly drawn with replacement, i.e., each entry can occur more than once (also called bootstrapping). Then, on each subsample, we perform the same process but column-wise (feature-wise). Afterward, for each subsample, we train an independent decision tree. We keep creating new trees until we reach the maximum number of trees. In the end, whenever we need to make a prediction, we take the prediction of each tree as a vote, look for the one with the most votes, and make it our final prediction. The probability value would be equal to the number of votes of the positive class divided by the number of trees.

A Random Forest contains several hyperparameters to optimize. For instance, we would look for the optimal number of trees. Also, we can look for the best number of features to be used at subsampling. We can look for other decision tree hyperparameters like the maximum depth of a tree or the maximum number of leaves in a tree.

III.2.5. Neural Networks

A neural network, as the name suggests, is a deep learning technique inspired by the biological brain. They are powerful in discovering patterns and solving complex problems that traditional machine learning cannot solve. They can be used for image-related problems, text-related problems, sound-related problems, time-series forecasting, and many more. Also, they can be used for regression as well as classification.

The most basic neural network is called a feedforward neural network (Multi-Layer Perceptron) [Ros58]. Feedforward neural networks consist of multiple layers that are connected. Layers can be divided into three groups. Input layers which come on first in the network and take on different forms of input. Output layers which come on last and give the final results in a network (such as a classification or prediction). Finally, we have hidden layers (also called hidden state) that come in between the input and output layers. Each of those layers consists of multiple neurons. Figure III.2 illustrates a simple feedforward network.

Neurons from one layer are connected to neurons from the previous layer. Equation III.10 explains how a value in a neuron is calculated. For instance, a neuron i from layer k is calculated using different neurons from the previous layer $k-1$. Also, neuron i holds a weight w_{ji}^{k} to scale the value of neuron j from the previous layer $k-1$. Usually, the resulting value is transformed to a 0-1 range using an activation function, mostly a sigmoid (σ) function (equation III.11). This sigmoid function transforms negative values to the range $(0-0.5)$, the positive values to the range of $(0.5-1)$, and zeros to 0.5. In general, any value o_j^k changes with change in input, and any weight w_{ji}^k or bias b_i^k is learned through training. This means, once we finish our training, weights and biases are fixed.

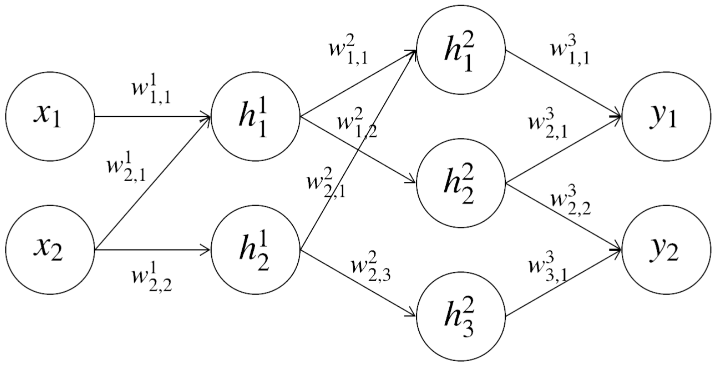


Figure III.2. A Feedforward Neural Network.

Neural networks use an optimizer for their training. This optimizer enhances performance by lowering the value of the loss function. The loss function is a measure of the inaccuracy of the model. It goes high whenever the model is inaccurate and goes low if the model is doing a good job. Most optimizers rely on the idea of a gradient. In our study, we are using an optimizer called Stochastic Gradient Descent (SGD) [Fri02]. SGD is an optimization algorithm that finds the local minimum of a function (loss function) using its gradient (slope). In neural networks, we represent our loss function in terms of neuron weights. It means that we are trying to find the optimal set of weights that minimizes our loss and boosts our performance.

SGD starts by plugging in random weight values to kick-off. Afterward, it calculates the gradient of the loss function. The gradient is calculated by taking the partial derivative for each weight at its current value. Then, it calculates a step size by scaling the slope with a learning rate. This learning rate helps the algorithm not to overshoot its step and miss the local minimum. Finally, the step size is used to update the current weight value. In general, this operation is done using each training input at once, and then averaging the step sizes from all of them. However, SGD picks up a subsample from the input and applies the optimization to it. Subsampling helps reduce the time and complexity of the learning process. In the learning process, we call this a step in the network. A group of steps over a batch of train samples is called an epoch. The algorithm continues to learn until it is satisfied with the loss or the maximum number of epochs or steps is reached.

$$h_i^k = b_i^k + \sum_{j=1}^{n_{k-1}} o_j^{k-1} \cdot w_{ji}^k \tag{III.10}$$

$$o_i^k = \sigma(h_i^k) = \frac{1}{1 + e^{h_i^k}} \tag{III.11}$$

III.2.6. Recurrent Neural Networks (RNNs)

RNNs are networks that perform sequential data modeling [Fau94]. They work on tasks that need context and information from previous time steps. RNNs can perform various tasks such as language translation, speech recognition, and stock prediction, which traditional neural networks cannot handle greatly. This is because traditional networks do not deal with sequential data contextually. RNNs can memorize information from earlier time steps and forward it to later time steps. Thus, maintain the context and the temporal relationship of inputs.

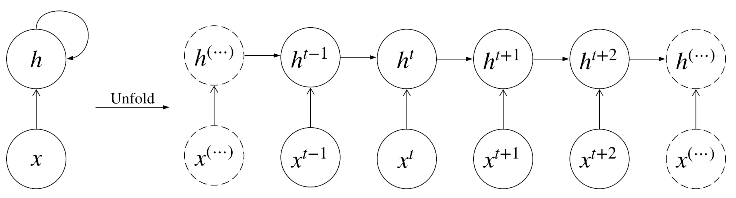


Figure III.3. A Recurrent Neural Network.

An RNN adds a loop to the hidden state $\mathbf{h}^t$ of a neural network (figure III.3 and equation III.12). The loop makes the output of a state become another input for it in the upcoming training step. A hidden layer ends up having two inputs, the traditional one which describes the current time step, and the input from its previous state that corresponds to all previous time steps. When a hidden layer receives those two inputs, it multiplies them by learned weights W, it concatenates them in a vector, and then it applies a hyperbolic tangent function ($tanh$) to each entry within that vector. The $tanh$ function squishes all the values between -1 and 1. The function ensures that the values do not explode over time. However, as RNNs go further in processing the sequence, the earlier time steps become less significant and have a negligible contribution to the most recent learned hidden state. This issue is known as the short-memory problem. In the next two subsections, we are going to explain two variations of RNNs that address and mitigate this problem.

$$\mathbf{h}_t = \begin{cases} 0, & t = 0 \\ tanh(W^{hh}h_{t-1} + W^{hx}x_t), & t > 0 \end{cases} \tag{III.12}$$

III.2.7. Long Short-Term Memory (LSTM)

LSTMs are advanced RNNs that can handle the short-memory [GSC99]. In particular, they can retain information from earlier time-steps if they are meaningful to the context. They introduce the concept of gates. Gates help to coordinate what to remember, what to forget, and what to pass to the next state. LSTMs consist of four main components; a cell state, a forget gate, an input gate, and an output gate. An LSTM unit is illustrated in figure III.4 and the formulae in equation III.13.

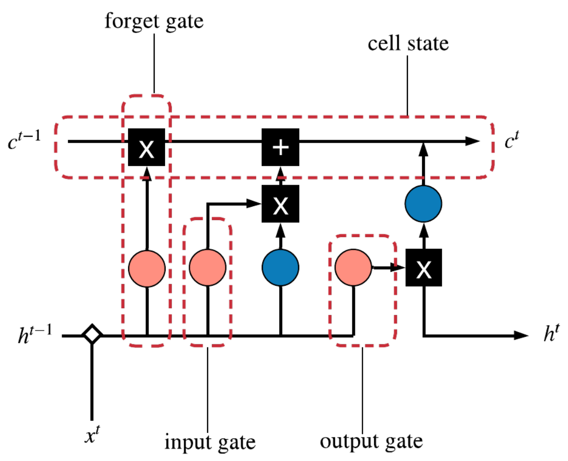


Figure III.4. An LSTM Unit.

The cell state acts as a transport highway that transfers relevant information to the rest of the sequence chain. Information from earlier time steps can be carried to the last time step, which reduces the effect of short term memory. Information gets added or removed via gates. They contain sigmoid activations which help in keeping information (closer to 1) or forgetting them (closer to 0). The forget gate $\mathbf{f}^t$ decides which information to throw away. The previous hidden state $\mathbf{h}^{t-1}$ and the current input x^t are concatenated in a vector. Afterward, they are passed to the sigmoid (red circle) as a vector to figure out what to keep or leave. The output of the sigmoid is then multiplied with the previous cell state $\mathbf{c}^{t-1}$. The input gate $\mathbf{i}^t$ updates the cell state $\mathbf{c}^t$. Again, we pass the previous hidden state and the current input to a sigmoid. Then, we pass the result to a *tanh* (blue circle) to decide what to update and how much to update. Then, it adds the new vector to the vector modified vector of the previous cell state, which produces the current cell state. The output gate $\mathbf{o}^t$ decides what the next hidden state $\mathbf{h}^t$ is going to be. It puts the current cell state through a *tanh* then it multiplies the output with the same sigmoid output of the forget gate. Again, W values refer to weights at each operation in the unit.

$$\begin{aligned}
\mathbf{i}^t &= \sigma(W^{ix}x^t + W^{ih}h^{t-1}) \\
\mathbf{f}^t &= \sigma(W^{fx}x^t + W^{fh}h^{t-1}) \\
\mathbf{o}^t &= \sigma(W^{ox}x^t + W^{oh}h^{t-1}) \\
\mathbf{g}^t &= tanh(W^{gx}x^t + W^{gh}h^{t-1}) \\
\mathbf{c}^t &= c^{t-1} \odot f^t + g^t \odot i^t \\
\mathbf{h}^t &= tanh(c^t) \odot o^t
\end{aligned} \tag{III.13}$$

III.2.8. Gated Recurrent Units (GRUs)

GRUs are another variant of RNNs with gates [CVMG+14]. GRUs are similar to LSTMs. They both address the issue of short-memory. However, GRU solves it in a different way that is computationally less complicated and more efficient [CGCB14]. GRUs get rid of the cell state and use the hidden state to transfer information.

On the contrary, it has two gates only, an update gate, and a reset gate. The update gate decides how much of the previous state should matter now. Moreover, the reset gate is used to drop out information the is irrelevant in the future. Figure III.5 defines a GRU unit and the different components of it along with the formulae in equation III.14. Here, $\mathbf{z}_t$ and $\mathbf{r}_t$ are update and reset gates respectively, and $\tilde{\mathbf{h}}_t$ is the candidate activation/hidden state.

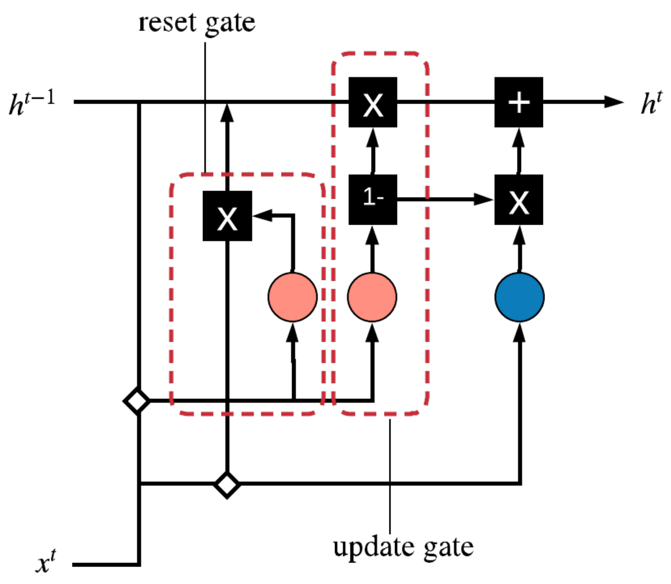


Figure III.5. A GRU Unit.

$$
\begin{aligned}
\mathbf{z}^t &= \sigma(W^{zx}x_t + W^{zh}h^{t-1}) \\
\mathbf{r}^t &= \sigma(W^{rx}x_t + W^{rh}h^{t-1}) \\
\tilde{\mathbf{h}}^t &= tanh(W^{x}x_t + r_t \odot W^{h}h^{t-1}) \\
\mathbf{h}^t &= z^t \odot h^{t-1} + (1 - z^t) \odot \tilde{h}^t
\end{aligned}
\qquad \text{(III.14)}
$$

CHAPTER IV
METHODOLOGY

Our overall approach towards attack detection / prediction is shown in Figure IV.1. The approach can be distinctly separated into the following stages — 1) Data collection $\rightarrow$ 2) Data Engineering $\rightarrow$ 3) Model Development $\rightarrow$ 4) Model Evaluation. In the data collection stage, we first collect our convert our data to proper formats. Then, we drop the packets/connections that are missing some attributes. For data engineering, we start by feature engineering. We do it by aggregating our attributes using a sliding-window fashion to generate features on the window level. Then, we perform our label engineering by assigning appropriate labels based on the purpose of the label (detection or prediction). For the model development and evaluation, we create our detection models at the beginning, evaluate, and analyze them. Based on the detection models' development and evaluation, we develop the prediction neural network models and evaluate them. Then, based on those, we create a combined dataset and develop a new model then evaluate it.

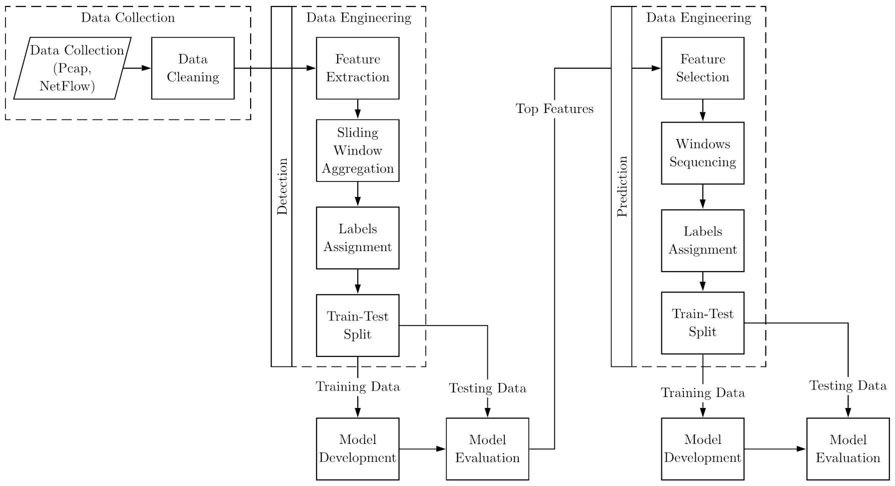


Figure IV.1. Our Methodology's Flow Diagram.

IV.1. Data Engineering

Data engineering stands out as of the most critical phase in the machine learning process. It hugely affects the behavior and performance of a machine learning model. It includes feature extraction, label extraction, and the manipulation of both. This section elaborates on explaining this process for the features and labels independently.

IV.1.1. Feature Engineering

In our study, feature engineering is divided into two sequential phases. The first is feature extraction, and the second is feature manipulation. Feature extraction aims to create feature-sets out of the raw data. On the other hand, feature manipulation takes the feature-set and applies different transformations over it, such as scaling, crossing, encoding, and so.

As mentioned before, the dataset comes in two formats; Pcap and NetFlow. This required creating two separate feature-sets. However, both of them have several features in common. The Pcap feature-set represents features on the packet-level, and the NetFlow feature-set represents features on the connection-level. Each one has its own advantage and unique features.

Feature extraction went into two phases. First, we create multiple time windows of equal length (0.1, 1, 10 seconds) and annotate the packets or connections into the one or more windows depending on if they were carried on during that time window. Second, we aggregate the data in each time window to create over 70 features for our models. Those features include counts, entropy values, mean values, and standard deviation values.

For both feature-sets, as shown in Table IV.1 and IV.2, we extracted 41 features for Netflow and 31 features for Pcap. A count refers to the number of packets/connections meeting a certain condition. The entropy, as described in equation III.3, is used to reflect on the distribution of values in a nominal field. The mean represents the arithmetic mean or average, which is the summation of a collection of a number divided by their count. The standard deviation reflects the variation in values for numerical fields. Based on those four measures, all of our features are numerical.

After extracting the features, and before feeding them to a model, we first apply standard scaling (also known as z-scoring). The second step is to apply the polynomial feature crossing. Finally, if the model supports sequential data, we group the data into sequences. Standard scaling is applied for each feature independently. It calculates the mean of the feature, as well as its standard deviation. Then, it creates a scaled representation of the collection by applying the formula in equation IV.1 to each feature value. This unifies the scale across all the features and helps in learning unified, making it easier for neural networks to learn all features at the same learning rate.

$$\mathrm{z}(x) = \frac{x - \bar{x}}{\sigma_x} \tag{IV.1}$$

Afterward, we apply a polynomial feature crossing of degree two. Polynomial crossing creates new features by multiplying the original features with each other. A degree of two indicates that we can multiply two features at most to create a new feature. Assuming we had n features, this would add $n \cdots (n-1)$ new features. This would make it easier for the algorithms to discover interactions between features.

Finally, if the model supports sequential data (recurrent neural network), we sequence the windows in each feature-set. Sequencing is the process of splitting ordered data into equal-width buckets. If the m^{th} window slot is n windows-wide, then the first slot (m_0) contains all the windows from $1 \cdots n$, the second slot (m_1) contains all the windows from $n + 1 \cdots 2n$, and so on. Sequences give the machine learning algorithm detailed information and a more profound context related to the state of the network. In our study, we experimented with different values of n, trying to optimize model performance.

IV.1.2. Label Engineering

Anomaly Detection: In order to create a supervised machine learning model capable of detecting attacks, we needed to annotate each connection/packet with its label. The CTU-13 dataset includes a list of IP addresses of malicious entities. Based on that list, we created two main classes — class 0 (normal) and class 1 (attack). A window is annotated as class 1 if an attack run during it, otherwise, it is annotated as class 0. The distribution of the those labels is shown in table IV.3.

Anomaly Prediction: While the detection labeled the current window to be in the different classes, the prediction model needed to be trained on future occurrences of the attack. For the purpose, we created different datasets based on the number of window steps (x) in the future we wanted to predict. For example, if the step size was $x = 1$, for the first slot the sequence of prediction feature vectors are $[v_p^1 <> \cdots v_p^{n-1} <>]$, which are labeled to be $l^n \in [0, 1]$ is the n^{th} window label. For $x \in [1, \cdots, 100]$ and $n \in [1, \cdots, 10]$, 1980 different datasets were created.

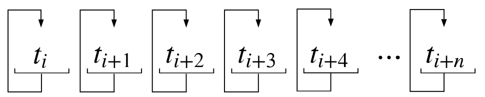


Figure IV.2. A 1-Window Sequence Used to Predict Self-State.

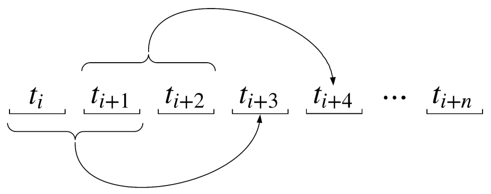


Figure IV.3. A 2-Window Sequence Used to Predict State 2 Steps Ahead.

After obtaining our best model, we further expanded the labeling to distinguish between the different attacks in the same dataset. The new set consists of four classes, which represent the presence of attacks over the IRC protocol — 0) No Attack, 1) DDoS, 2) Spam, and 3) General attacks on IRC.

Table IV.1. NetFlow Feature-Set.

Feature	Description
m_duration	Mean connection duration
n_conn	Count of all connections
n_normal	Count of normal connections
n_src_ip_a	Count of source IP by Class
n_src_ip_b	
n_src_ip_c	
n_src_ip_na	
n_dst_ip_a	Count of destination IP by Class
n_dst_ip_b	
n_dst_ip_c	
n_dst_ip_na	
n_src_port<1024	Count of ports over and below 1024
n_src_port>=1024	
n_dst_port<1024	
n_dst_port>=1024	
n_tcp	Count of protocol used
n_udp	
n_icmp	
ent_duration	Entropy - connection duration
ent_packets	Entropy - packets transferred
ent_srcbytes	Entropy - bytes from source
ent_state	Entropy - connection states
ent_totbytes	Entropy - bytes transferred
sd_duration	Standard deviation - duration per connection
sd_packets	Standard deviation - packets transferred
sd_srcbytes	Standard deviation - bytes from source
sd_totbytes	Standard deviation - bytes transferred
ent_src_ip_a	Entropy - source IP by class
ent_src_ip_b	
ent_src_ip_c	
ent_src_ip_na	
ent_dst_ip_a	Entropy - destination IP by class
ent_dst_ip_b	
ent_dst_ip_c	
ent_dst_ip_na	
ent_src_ip	Entropy - source and destination IP addresses
ent_dst_ip	
ent_src_port<1024	Entropy - source and destination ports
ent_src_port>=1024	
ent_dst_port<1024	
ent_dst_port>=1024	

Table IV.2. Pcap Feature-Set.

Feature	Description
n_packets	Count of all packets
n_normal	Count of normal packets
n_src_ip_a n_src_ip_b n_src_ip_c n_src_ip_na	Count of source IP by Class
n_dst_ip_a n_dst_ip_b n_dst_ip_c n_dst_ip_na	Count of destination IP by Class
n_src_port<1024 n_src_port>=1024 n_dst_port<1024 n_dst_port>=1024	Count of ports over and below 1024
n_tcp n_udp n_icmp	Count of protocol used
ent_src_ip_a ent_src_ip_b ent_src_ip_c ent_src_ip_na	Entropy - source IP by class
ent_dst_ip_a ent_dst_ip_b ent_dst_ip_c ent_dst_ip_na	Entropy - destination IP by class
ent_src_ip ent_dst_ip	Entropy - source and destination IP addresses
ent_src_port<1024 ent_src_port>=1024 ent_dst_port<1024 ent_dst_port>=1024	Entropy - source and destination ports

Table IV.3. Window Label Classes Distribution Among the Different Datasets. (NF: NetFlow, PC: Pcap)

	0.1-Second			1-Second			10-Second		
	DDoS	Spam	IRC	DDoS	Spam	IRC	DDoS	Spam	IRC
NF-0	1.6e4	2.2e5	364418	15200	9343	142179	1494	1150	15313
NF-1	1.05e6	27118	2.9e6	101341	23846	184311	10162	2170	17482
PC-0	2e5	2.08e5	366710	23654	18402	175980	1600	1593	18077
PC-1	1.01e6	59729	102098	92897	7603	153611	10058	894	14049

IV.2. Model Development

Our supervised learning process consists of two parts. The first is to detect attacks with machine learning methods. The second is to predict attacks with neural networks. Equation IV.2 illustrates our approach in general. For the machine learning, we pass a vector of detection features $v_d^i <>$, trying to map it to the label vector l^i using a learned function f. However, for neural networks, the feature vector $v_p^i <>$ is mapped to the label vector l^{i+x}, where $x \geq 1$. Also, $v_p^i <>$ consists of the ten most important features rather than all of them as in $v_d^i <>$.

$$< v_{[d/p]} > \xrightarrow{f(v)} l_{[0/1]} \tag{IV.2}$$

IV.2.1. Machine Learning Models - Detection

For our machine learning models, the design of a model was limited to hyperparameter tuning for tree-based models. For our Random Forest and Gradient Boosting, we fed the Bayesian optimization search algorithm with hyperparameters, and it trained the model on different combinations of them until it found the best one. Table IV.4 shows our hyperparameters for Random Forest and Gradient Boosting.

IV.2.2. Neural Network Architectures - Prediction

In order to develop our sequence based prediction model, we leverage the base LSTM/GRU models (as described in Section III) and build a multi-feature pipeline architecture. Figure IV.4 shows the general architecture of the model, where the models is able to take 10 different inputs passed through a series of LSTM and GRU units. The following equation describes the transformation and training of the v_p sequence feature vector to map towards the attack scenarios. Where, ϱ is the LSTM/GRU units which take the a sequence of a single feature $v_p^0, \cdots, v_{p0}^t$ independently. Each LSTM/GRU unit learns a feature sequence, then the output from all the units is concatenated in V. Afterward, V is fed to a sigmoid function that maps it to the final label l^{t+x} to conclude the model. The model utilizes SGD with a learning rate of 0.001. This value was reached after several experiments with a range of values between (0.0001 - 1).

Table IV.4. Random Forest and Gradient Boosting Hyperparameters.

Parameter	Values Tested
No. of Trees	10,50,100,200,500
Min. Samples Split	100-1000
Max. Tree Depth	5-15
Min. Samples per Leaf	100-1000
Learning Rate (Gradient Boosting only)	0.01-1

$$\left.\begin{aligned} \varrho(v_{p0}^0, \cdots, v_{p0}^t) &\rightarrow \mathbf{h}_0^t \\ \varrho(v_{p1}^0, \cdots, v_{p1}^t) &\rightarrow \mathbf{h}_1^t \\ &\vdots \\ \varrho(v_{p9}^0, \cdots, v_{p9}^t) &\rightarrow \mathbf{h}_9^t \end{aligned}\right\} \oplus \rightarrow V \rightarrow \sigma \rightarrow l^{t+x} \tag{IV.3}$$

IV.2.3. Training

After we put the data through a preprocessing pipeline that implements the data engineering process, we split our data into train and test subsets. Our training subset contains 70% of the original dataset, and the testing is done on the rest 30%. For each of machine learning techniques, we train on detecting all of the three types of attack individually using different window lengths and different feature-sets. As a result, the total number of models evaluated is 54. On the other hand, for our neural network models, we train a total of more than 2000 models split up between the different attacks, feature-sets, and recurrent neural network types.

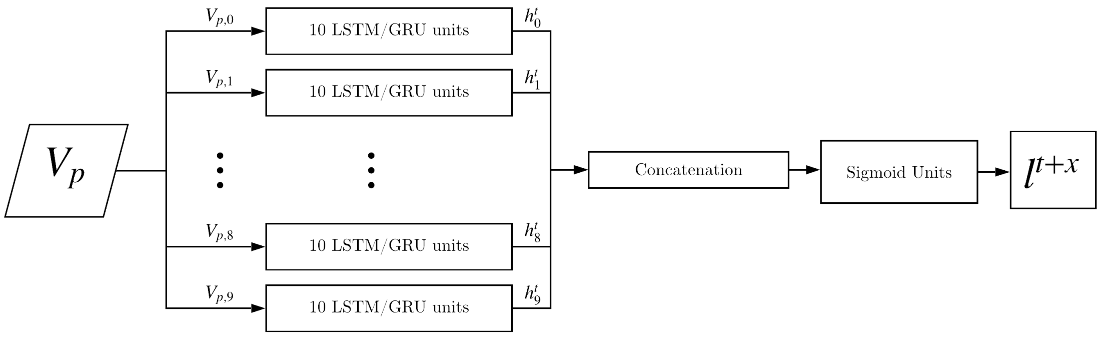


Figure IV.4. Our Neural Network Architecture.

For each model, we record results over 10-fold cross-validation. The optimization of the model hyper-parameters is tightly coupled within the cross-validation stages. For Random Forest, and Gradient Boosting models, the hyper-parameters are evaluated via a Bayesian approach towards optimization [PGCP99]. The approach is based on developing a probability-based objective function, which keeps historical track of the model $f()$ metrics obtained (via various sets of hyper-parameters) in order to get closer to the optimal set.

IV.3. Evaluation

Evaluating the developed models was done for two primary reasons, first to judge the performance and reliability of the model and second, to establish a common ground for cross-evaluation/comparisons between the models.

In our study, we utilize multiple evaluation criteria (as shown in Table IV.5), to understand the performance metrics of the developed models. Generally, we use precision, recall, and F1-score as our standard evaluation matrices, with Kolmogorov-Smirnov (KS) test statistics value and area under the receiver operating characteristic curve (AU-ROC) enabling an analysis of probability cutoff thresholds. Precision describes the fraction of True Positives (TP) against all the positive predictions, recall gives the fraction of TP against all the actual positives, and F1-score combines both precision and recall into a more generalize score. While precision indicates how much we can trust a positive prediction by the model, recall suggests how confident are we in being able to predict actual positives. Both precision and recall are useful metrics and performance estimates, but sometimes we might want to compare or evaluate based on both of them combined, this is where F1-score comes into the picture. F1-score is defined as the harmonic mean of both recall and precision, where it averages both metrics and gives an overall indication of the precision-recall performance.

These metrics are calculated based on a confusion matrix. A confusion matrix in a classification problem is a $c \times c$ matrix, where c is the number of classes. This matrix, as shown in Figure IV.5, represents the actual classes along the rows and the predicted classes along the columns, or vice versa. The top left entry represents the true negatives (TN), the top right entry shows the false positives (FP), the bottom left entry gives us the false negatives (FN), and the last one gives us the true positives (TP).

As each model gives us a probability, we need a probability threshold value to create our confusion matrix. Usually, the threshold is set to 0.5; however, we can use the KS test statistics value to determine the best threshold. In particular, we can create a curve for each of the negative and positive classes. Those two curves are created by plotting the threshold against the percentage below that threshold. Then, we look for the threshold with the maximum vertical distance between the two curves. This threshold is guaranteed to have the best F1-score among all the thresholds. Finally, the ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR). Different pairs of TPR and FPR are calculated by varying the threshold. After that, we can calculate the area under that curve to get the AU-ROC value on a scale from zero to one, with one representing a perfect classifier.

Table IV.5. Evaluation Metrics.

Metric	Formula
Precision	$\frac{TP}{TP+FP}$
Recall/TPR	$\frac{TP}{TP+FN}$
F1-Score	$2 \cdot \frac{recall \cdot precision}{recall+precision}$
KS Test Statistics	$\sup(F_{C1} - F_{C2})$
FPR	$\frac{FP}{FP+TN}$

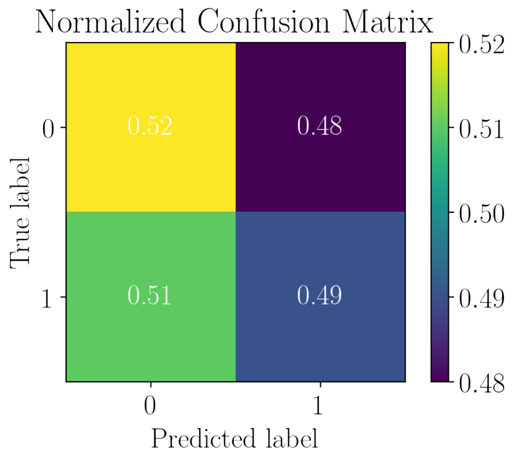


Figure IV.5. A Confusion Matrix.

IV.4. Interpretability

In order to gain a deeper understanding of the developed models, it is essential to explore the feature to detection/prediction decision relationship. This includes understanding what features are being used by the model and what features are being dismissed. Also, the questions of — the effect each feature have on a prediction; does it move the prediction towards one class or the other? How does the range of values in a feature affect the prediction? can be examined further. Interpretability assures us that the model is not basing its decision on irrelevant information or contexts. Besides, it makes it easier to troubleshoot the data engineering process and the training process.

Within our approach, we primarily used two methods to explain the model behavior. The first one is Pearson's correlation coefficient (equation IV.4). Pearson's correlation is a statistical test that measures the relationship or association between two numerical features. In particular, it is used to measure the correlation between the different features and prediction probabilities. This approach assumes independence between the two features and that the correlation between them is linear. Also, it describes the correlation by a real value from -1 to 1, with -1 describing a perfect negative correlation, 1 describing a perfect positive correlation, and 0 describing the absence of a correlation. A negative correlation means that if one of the features increases, the other decreases to a certain degree. A positive correlation means that both features increase and decrease together.

$$P_{X,Y} = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y} \tag{IV.4}$$

On the other hand, we have used SHapely Additive Explanations (SHAP) [LL17]. SHAP unifies several model interpretability approaches, LIME [RSG16], DeepLIFT

[SGK17], and Shapley [ŠK14] are the mainly used ones. SHAP can interpret traditional machine learning models such as Random Forest and Logistic Regression, but it also can understand and interpret complex neural networks. Although SHAP assumes co-linearity and features independence, it extends the concept much further. SHAP interpolates new data from a given dataset, and experiments with removing (zeroing) a specific feature at a time and keeping the rest. Interpolation is done on each feature individually to generate new values that still relate to the nature of the feature. Besides, zeroing a specific feature allows for a better understanding of its importance based on the change in model behavior when it is absent.

CHAPTER V
RESULTS AND DISCUSSION

In this chapter, we show, analyze, and interpret the results obtained with the different detection and prediction models used in this study. A comprehensive comparison is presented to highlight the best approach, along with the best settings.

V.1. Detection

Traditional machine learning techniques were used to detect attacks. Table V.1 shows the F1-scores for attack detection with Netflow feature-sets. We observe that for DDoS attacks, Random Forest and Gradient Boosting do well, while Logistic Regression is performing poorly. For DDoS with a 0.1-second window, we can notice that Random Forest comes performs a little bit better than Gradient Boosting. However, with a 1-second window, Gradient Boosting shows great performance and a noticeable difference compared to Random Forest. The best performance is observed in a 10-second window, where Gradient Boosting maintains the best performance with nearly a perfect score. Also, spam attacks are rather not predicted with Logistic Regression. With a 0.1-second window, Random Forest still performs better than Gradient Boosting and Logistic Regression. Though, Gradient Boosting shows a 0.06 better performance than Random Forest with a 1-second window. Utilizing a 10-second window, Gradient Boosting still detects spam attacks better than the other models. Finally, other attacks over IRC are always detected best by Gradient Boosting. In the 0.1-second window scenario, they have a small advantage over Random Forest and a massive advantage over Logistic Regression. A 1-second window and a 10-second window are still dominated by Gradient Boosting ahead of the two other approaches with 0.99 scores. Thus, Gradient Boosting proves to be the best model to detect attacks with a 10-second window Netflow feature-set.

Table V.1. F1-Score for Attack Detection using Netflow Data. (LR: Logistic Regression, RF: Random Forest, GB: Gradient Boosting)

	0.1-Second			1-Second			10-Second		
	DDoS	Spam	IRC	DDoS	Spam	IRC	DDoS	Spam	IRC
LR	0.401	0.469	0.486	0.388	0.465	0.519	0.637	0.697	0.677
RF	0.918	0.938	0.938	0.951	0.927	0.927	0.958	0.922	0.941
GB	0.913	0.935	0.953	0.990	0.989	0.992	**0.999**	**0.999**	**0.999**

On the contrary, machine learning performance with the Pcap feature-set is illustrated in table V.2. For DDoS detection with a 0.1-second window, we notice a massive jump in Logistic Regression performance from 0.401 with Netflow to 0.989 with Pcap. However, Random Forest and Gradient Boosting are still performing equally good, and better than Logistic Regression. The three models drop in performance with a 1-second window, and Gradient Boosting goes back at the top followed by Logistic Regression. Moreover, a more significant drop in Logistic Regression and Random Forest performance is observed with a 10-second window. Gradient Boosting held the best performance for that window along with the best DDoS overall detection performance. For spam attacks, Gradient Boosting stays ahead of Random Forest by 0.13, with logistic being a little bit behind of Random Forest. Pcap with a 1-second window maintains the same overall performance as the 0.1-second window, with Gradient Boosting achieving the best overall spam detection performance with this window size. A 10-second window nearly matches the performance of the previous window size with Gradient Boosting at the top. Lastly, other IRC attacks can be detected a little bit better than spam. With a window of 0.1-second, Logistic

Regression and Random Forest show some improvement, but Gradient Boosting leads in performance. A longer window of 1-second gets a perfect score with Gradient Boosting, while other models stayed behind. The 10-second window shows some degradation in Gradient Boosting performance, but it maintained the best performance as usual.

Table V.2. F1-Score for Attack Detection using Pcap Data.

	0.1-Second			1-Second			10-Second		
	DDoS	Spam	IRC	DDoS	Spam	IRC	DDoS	Spam	IRC
LR	0.989	0.848	0.898	0.929	0.844	0.847	0.873	0.848	0.901
RF	0.995	0.866	0.938	0.925	0.867	0.886	0.844	0.866	0.911
GB	0.995	0.993	0.995	0.971	**0.995**	**1.000**	**0.997**	0.993	0.945

In general, Gradient Boosting performed better across most of the attack-window size combinations. All of the attacks were nearly perfectly detected by Gradient Boosting using both feature-sets. However, Pcap holds the best performing model; Gradient Boosting for IRC attacks detection, and Netflow holds the worst performing model with Logistic Regression DDoS detection. The difference between the two feature-sets in Logistic Regression can be related to how granular a packet can be compared to a connection.

V.2. Detection - Feature Anaylsis

Our best performing models are then analyzed furthermore to interpret their behavior and extract their top features. Those are Gradient Boosting over Netflow with 10-second window to detect DDoS and Gradient Boosting over Pcap with a 1-second window to detect IRC attacks. Figure V.1 illustrates the top 10 features for our best Netflow model. The figure can be interpreted as the following; for a certain feature, dots on the positive side mean that the feature pushes the model to detect an attack. Dots on the negative side mean that the feature pushes the model to detect a normal behavior. If the dots are red, then higher values of that feature push in that direction. If the dots are blue, then lower values of that feature push the model towards that direction.

The most important feature in that model is **n-src-ip-a**. The figure indicates that a higher number of connections with source IP addresses from class A lead to attack detection, while a lower number confirms the absence of an attack. Our next important feature is **ent-dst-ip-c**. It refers to the entropy in destination IP addresses in class C. A high variation in IPs from this class leads the model for attack detection. While lower variation barely affects the decision of the model. The next one is **ent-state** with a similar interpretation as the previous one. A variation in the connection state pushes the model a decision to attack detection, while the lower variation is hardly pushing the model against that. However, for **n-tcp**, a lower number of TCP connections lead to a negative decision of an attack absence, without a higher number leading to attack detection. The rest of the features can be interpreted the same way as well; however, their impact on the model is lower compared to the top four.

For our best Pcap model, figure V.2 depicts the top 10 features. **ent-dst-ip-c** comes up as the most important feature. The interpretation differs here as we are looking at the entropy of class C IPs on the packets level rather than the connection level. The second important feature is **ent-src-ip**. It can be interpreted as the following; a high variation in source IP addresses indicates an attack while a very low variation in source IP addresses indicates the absence of an attack. The next important feature is **ent-dst-port-lt-1024**. The figure indicates that a high variation in the used well-known destination ports leads the model to detect an attack, while lower variation pushes the model a little bit towards denying a detection. Most of the features are directly proportional to attack detection. However, The 9^{th} most important feature, **ent-src-port-lt-1024**, leads the model to deny an attack if the variation of well-known source port numbers high. A slightly pushes the model to detect an attack with lower values.

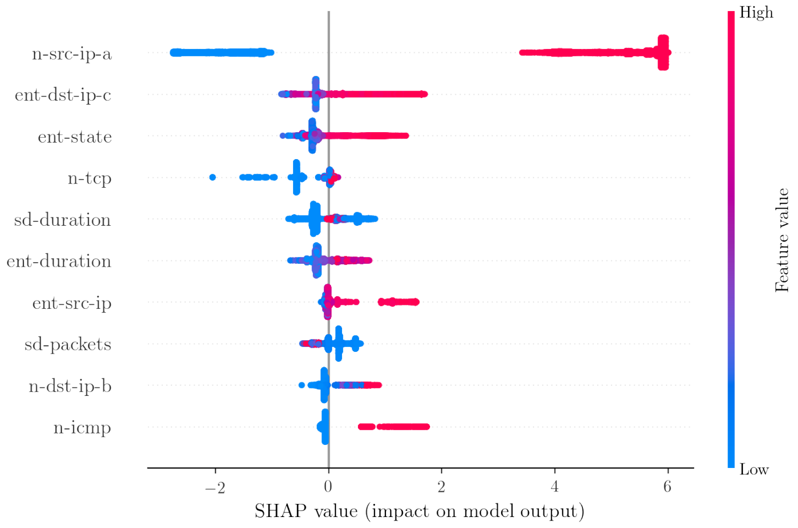


Figure V.1. Best Netflow Model - Top 10 Important Features.

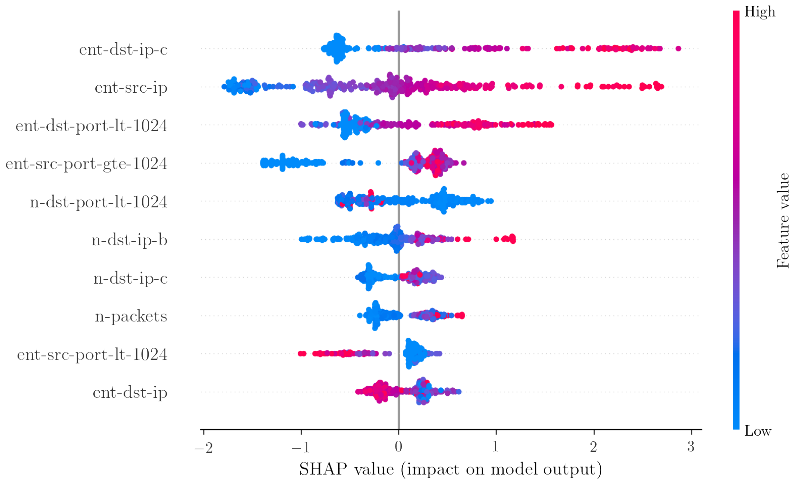
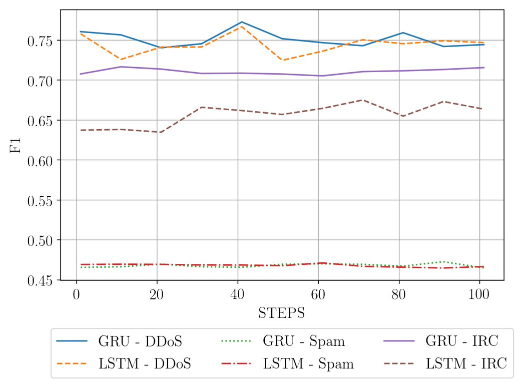


Figure V.2. Best Pcap Model - Top 10 Important Features.

V.3. Prediction

Our neural networks were used to predict attacks. Figures V.3 to V.8 illustrate the performance of GRU and LSTM networks in predicting attacks. Each couple of figures refer to window size and a feature-set combination. Each couple represent 660 different models for each couple, and all of them represent 3960 models in total. The ones on the left depict the change in F1-score with respect to different future steps. The ones on the right represent the change in F1-score with respect to the different sequence lengths. Figures V.9 and V.10 illustrate show the averaged performance on the feature-set level.

As shown in figure V.3, Netflow feature-set with 0.1-second window illustrated a poor but stable performance across different future steps as well as sequence lengths. DDoS is the best predicted attack followed by IRC attacks with a 0.76 F1-score. While spam attacks are poorly predicted by any model. In most cases, GRUs are slightly performing better than LSTMs for DDoS, notably performing better in IRC attacks, and nearly performing the same in spam attacks prediction.



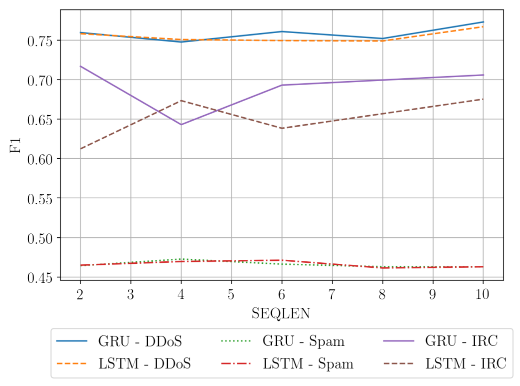


Figure V.3. Netflow 0.1-second Window RNNs Performance

For our Netflow 1-second window, shown in figure V.4, we notice a little bit of performance irregularity with respect to steps. However, the performance still degrades predicting farther steps into the future. DDoS is still the best predicted attack ahead of IRC attacks. Although spam prediction is behind, spam attacks are predicted better with this window size. Besides, GRU and LSTM models are performing at the same level in this case. In general, this dataset does its best with a step of 1 and a sequence length of 6 with an F1-score of 0.76.

The final netflow feature-set of 10-second windows is shown in figure V.5. This window is performing much better than the other two windows at a top value of 0.92 with a step of 1 and a sequence length of 6. Also, we can see a clear degradation in performance going farther in the future and a less clear degradation with extended sequence lengths. In general, DDoS became the worst predicted attack in this feature-set with a value of 0.71 at best. Spam became the best predicted attack with 0.92 score followed by IRC attacks at 0.82. Both GRU and LSTM performed the same.

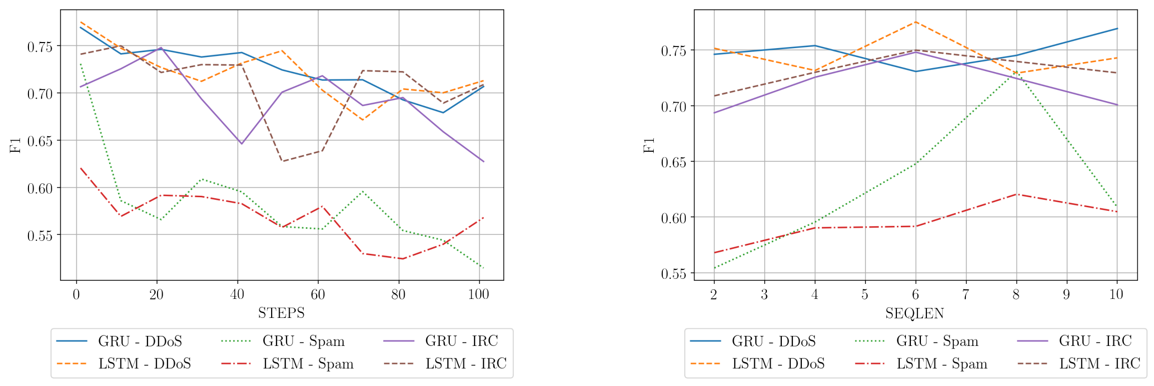


Figure V.4. Netflow 1-second Window RNNs Performance

On the other hand, Pcap scored high performance of 0.92 with 0.1-second window feature-set. The performance degraded a little bit predicting steps ahead, and nearly stayed the same with different sequence lengths. DDoS was the best predicted attack by a huge difference with a score of 0.92. IRC comes next with with values around 0.69 and spam with values around 0.5. In this feature-set, GRUs performed a slightle better than LSTMs. Pcap's performance is better than netflow's here.

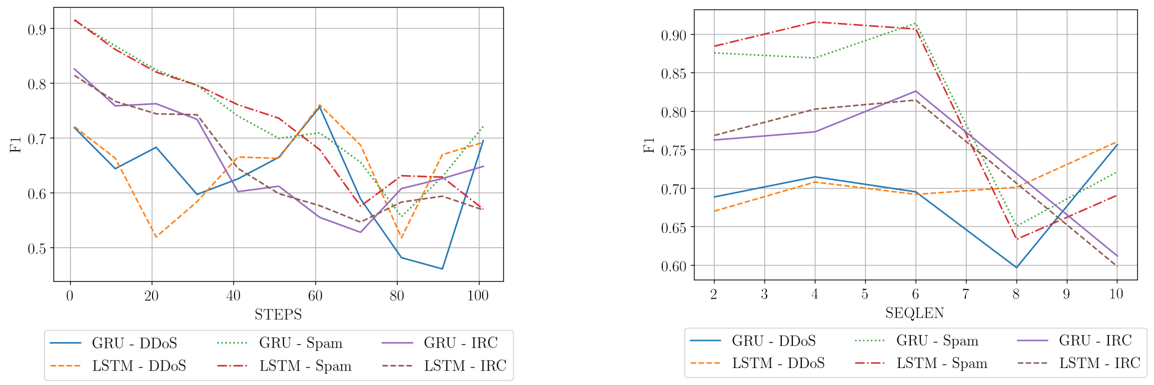


Figure V.5. Netflow 10-second Window RNNs Performance

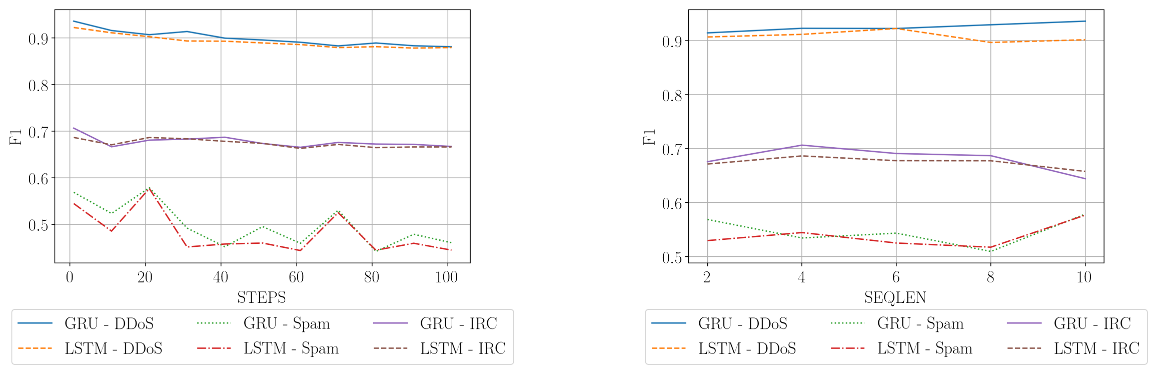


Figure V.6. Pcap 0.1-second Window RNNs Performance

A 1-second window in Pcap performed even better than the previous one. A high-score of 0.95 was recorded at 1 step and a sequence length of 8. The performance still degraded with time steps, but improved a little bit with sequence lengths. IRC attacks are predicted the best with this feature-set, then comes DDoS followed by spam attacks. GRUs and LSTMs nearly performed the same way with this feature-set.

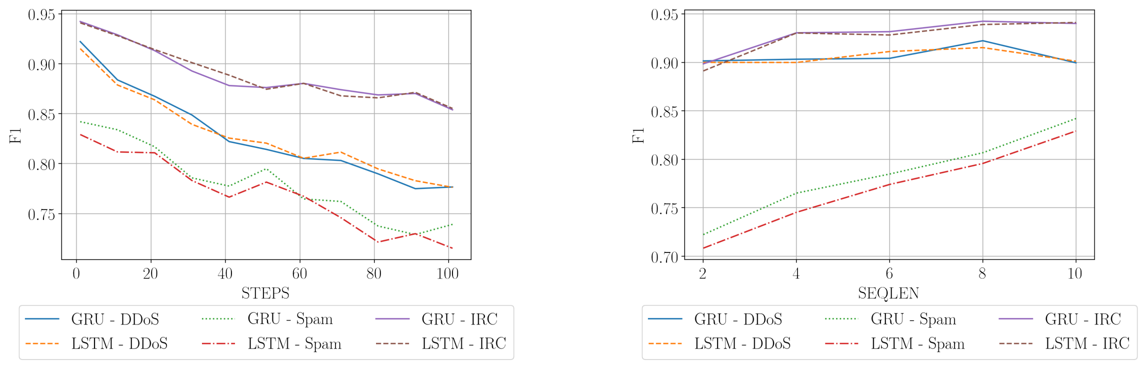


Figure V.7. Pcap 1-second Window RNNs Performance

Lastly, Pcap's 10-second window feature-set performed its best at 1 step and a sequence length of 2. Also, the performance still degraded notably going deeper into the future, but mostly stabilized with different sequence lengths. Once again IRC was the top predicted, followed by spam's highest recorded prediction performance. GRUs led LSTMs in different attack predictions.

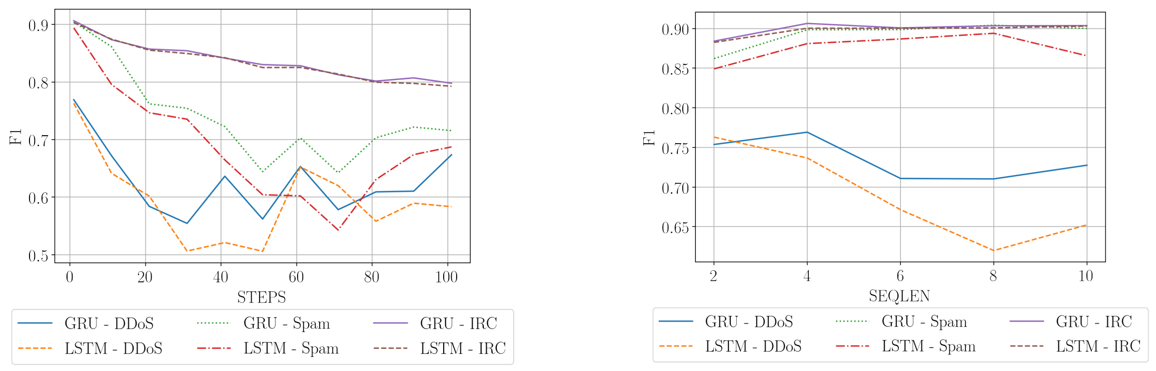


Figure V.8. Pcap 10-second Window RNNs Performance

In general, figure V.9 represents the overall performance of models fed with Netflow feature-sets. We can notice that the average F1-score degrades clearly with time steps and sequence lengths. We observe that GRU is performing slightly better on average and has the highest average of 0.67 at 1 step and a sequence length of 2.

On the other hand, V.10 illustrate the overall performance of models fed with Pcap feature-sets. We observe higher averages than Netflow's with a peak of 0.81 F1-score. The performance degrades with time-steps and sequence lengths, which leads to 1 step and a sequence length of 2 being the best combination. GRUs still performed notably better than LSTMs.

In our final comprehensive model (GRU with Pcap feature-set of 1-second windows), we look for AU-ROC scores. This illustrates how good are we predicting positive classes, and makes it easier for per class performance illustration. Figure V.11 shows the AU-ROC performance of GRU for predicting all kinds of attacks in one dataset. We can notice that the AU-ROC curve degrades by going farther in the future. However, the performance was reaches its best with a sequence length of 10 windows and degrades afterward. The model predicted DDoS better than the rest of the attacks, followed by spam, and ending up with other IRC Attacks.

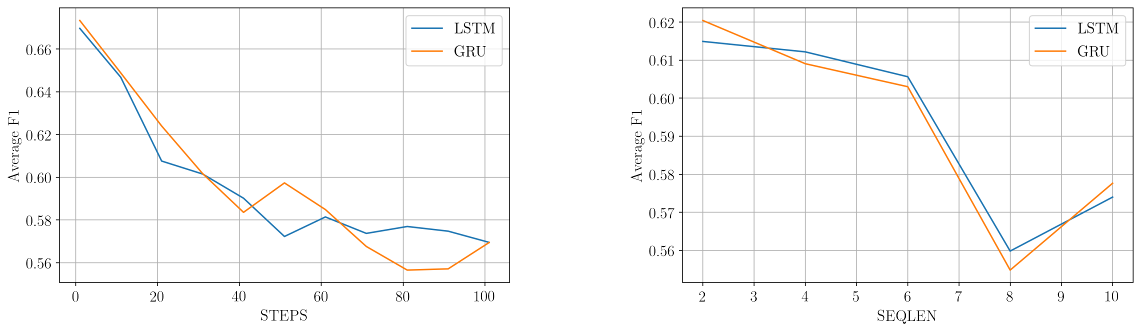


Figure V.9. Netflow Overall RNNs Performance

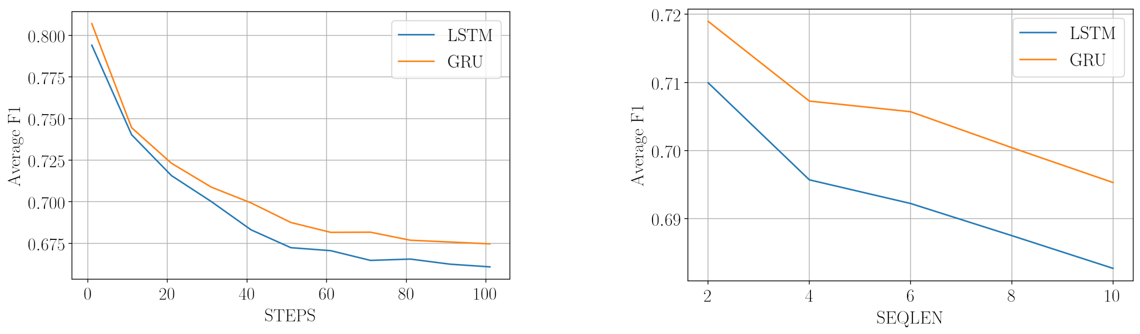


Figure V.10. Pcap Overall RNNs Performance

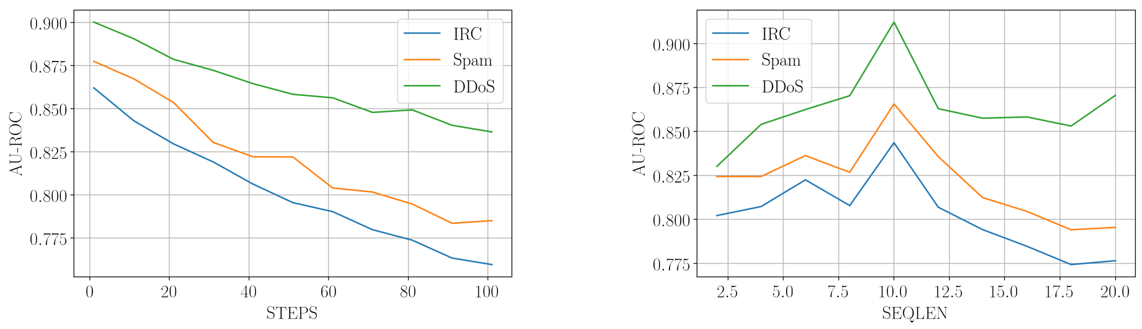


Figure V.11. GRU Pcap 1-second Window Performance

CHAPTER VI
CONCLUSION AND FUTURE WORK

In this thesis, we explored the performance of machine learning and neural networks in identifying malicious network behavior. In the beginning, we monitored the performance of the traditional machine learning techniques in detecting such behavior. We established a comparison between the different techniques that ended up with excellent attack detection performance with scores ≥ 0.99. Afterward, we picked up the best model for each feature-set and analyzed it further. The analysis ended up with interpreting the models and selecting the most important features to build upon in the next step.

Then, using the top features, hundreds of datasets created to cover a wide range of settings. This covered the different data formats, sliding window sizes, future steps, sequence lengths, and attack types. As a result, hundreds of neural network models, that utilize LSTMs and GRUs, were trained to predict the malicious behavior in a network. Afterward, the models were compared to choose a final data format with a sliding window size and neural network to be used in the next step.

Finally, we created a final dataset that combines all the different attacks. The best model from the previous step was trained on different sequence lengths and future steps and was evaluated after. The evaluation concluded with a sequence length of 10 and a future step of 1 as being the best combination for sequence building and modeling in our study.

This work can be extended to predict the rest of the attacks as well as other protocols. Also, more advanced and complicated neural network approaches may be potentially utilized for that purpose. It can also be expanded to feed the models with sequences of packets rather than window aggregations. Furthermore, it can be extended to predict the intensity/damage of an attack rather than the absence/presence of the attack and decide if it is worth blocking based on model confidence and cost prediction, which changes it from a classification problem to a regression problem.

Another approach may include directionally modeling the traffic rather than just including plain features about the source, destination, and direction, instead of leaving it up to the model to figure out what those features mean. Such an approach may lead up to a deeper understanding of the behavior and may potentially uncover hidden patterns.

Although there is room for exploration and improvement, always, we believe that our study covers many sides and provides a robust approach and strong models to model malicious network traffic.

BIBLIOGRAPHY

[AS18] Wafaa Anani and Jagath Samarabandu, *Comparison of recurrent neural network algorithms for intrusion detection based on predicting packet sequences*, 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), IEEE, 2018, pp. 1–4.

[Bre01] Leo Breiman, *Random forests*, Machine learning **45** (2001), no. 1, 5–32.

[Bre17] ———, *Classification and regression trees*, Routledge, 2017.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, arXiv preprint arXiv:1412.3555 (2014).

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078 (2014).

[DAF18] Rohan Doshi, Noah Apthorpe, and Nick Feamster, *Machine learning ddos detection for consumer internet of things devices*, 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 29–35.

[Fau94] Laurene Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*, Prentice-Hall, Inc., 1994.

[Fri01] Jerome H Friedman, *Greedy function approximation: a gradient boosting machine*, Annals of statistics (2001), 1189–1232.

[Fri02] ———, *Stochastic gradient boosting*, Computational statistics & data analysis **38** (2002), no. 4, 367–378.

[GGSZ14] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino, *An empirical comparison of botnet detection methods*, Computers & Security **45** (2014), 100–123.

[GSC99] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins, *Learning to forget: Continual prediction with lstm.*

[KKTK16] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim, *Long short term memory recurrent neural network classifier for intrusion detection*, 2016 International Conference on Platform Technology and Service (PlatCon), IEEE, 2016, pp. 1–5.

[LL17] Scott M Lundberg and Su-In Lee, *A unified approach to interpreting model predictions*, Advances in Neural Information Processing Systems 30 (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), Curran Associates, Inc., 2017, pp. 4765–4774.

[LWLS06] Carl Livadas, Robert Walsh, David E Lapsley, and W Timothy Strayer, *Using machine learning techniques to identify botnet traffic.*, LCN, Citeseer, 2006, pp. 967–974.

[PGCP99] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz, *Boa: The bayesian optimization algorithm*, Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1, Morgan Kaufmann Publishers Inc., 1999, pp. 525–532.

[Ros58] Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), no. 6, 386.

[RRD18] Benjamin J Radford, Bartley D Richardson, and Shawn E Davis, *Sequence aggregation rules for anomaly detection in computer network traffic*, arXiv preprint arXiv:1805.03735 (2018).

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, *"why should I trust you?": Explaining the predictions of any classifier*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, 2016, pp. 1135–1144.

[SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje, *Learning important features through propagating activation differences*, Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 3145–3153.

[ŠK14] Erik Štrumbelj and Igor Kononenko, *Explaining prediction models and individual predictions with feature contributions*, Knowledge and information systems **41** (2014), no. 3, 647–665.

[SP14] Matija Stevanovic and Jens Myrup Pedersen, *An efficient flow-based botnet detection using supervised machine learning*, 2014 international conference on computing, networking and communications (ICNC), IEEE, 2014, pp. 797–801.

[STG$^+$11] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian, *Detecting p2p botnets through network behavior analysis and machine learning*, 2011 Ninth annual international conference on privacy, security and trust, IEEE, 2011, pp. 174–180.

[XSDZ18] Congyuan Xu, Jizhong Shen, Xin Du, and Fan Zhang, *An intrusion detection system using a deep neural network with gated recurrent units*, IEEE Access **6** (2018), 48697–48707.

[ZZH08] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque, *Random-forests-based network intrusion detection systems*, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **38** (2008), no. 5, 649–659.