DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Anomaly Detection in Unstructured Time Series Data using an LSTM Autoencoder

MAXIM WOLPHER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ROYAL INSTITUTE OF TECHNOLOGY

Anomaly Detection in Unstructured Time Series Data using an LSTM Autoencoder

by

Maxim Wolpher

Examiner: Mads Dam
Advisor: György Dán

—

A thesis submitted in fulfillment for the degree of Master of Science in Engineering Physics
Master of Science, Engineering Physics

in the
School of Electrical Engineering and Computer Science

June 2018

Abstract

An exploration of anomaly detection. Much work has been done on the topic of anomaly detection, but what seems to be lacking is a dive into anomaly detection of unstructured and unlabeled data. This thesis aims to determine the effectiveness of combining recurrent neural networks with autoencoder structures for sequential anomaly detection. The use of an LSTM autoencoder will be detailed, but along the way there will also be background on time-independent anomaly detection using Isolation Forests and Replicator Neural Networks on the benchmark DARPA dataset. The empirical results in this thesis show that Isolation Forests and Replicator Neural Networks both reach an F1-score of 0.98. The RNN reached a ROC AUC score of 0.90 while the Isolation Forest reached a ROC AUC of 0.99. The results for the LSTM Autoencoder show that with 137 features extracted from the unstructured data, it can reach an F1 score of 0.8 and a ROC AUC score of 0.86

Sammanfattning

En undersökning av anomalitetsdetektering. Mycket arbete har gjorts inom ämnet anomalitetsdetektering, men det som verkar saknas är en fördjupning i anomalitetsdetektering av ostrukturerad och omärkt data. Denna avhandling syftar till att bestämma effektiviteten av att kombinera Recurrent Neural Networks med Autoencoder strukturer för sekventiell anomalitetsdetektion. Användningen av en LSTM autoencoder kommer att beskrivas i detalj, men bakgrund till tidsoberoende anomalitetsdetektering med hjälp av Isolation Forests och Replicator Neural Networks på referens DARPA dataset kommer också att täckas. De empiriska resultaten i denna avhandling visar att Isolation Forests och Replicator Neural Networks (RNN) båda når en F1-score på 0,98. RNN nådde en ROC AUC-score på 0,90 medan Isolation Forest nådde en ROC-AUC på 0,99. Resultaten för LSTM Autoencoder visar att med 137 features extraherade från ostrukturerad data kan den nå en F1-score på 0,80 och en ROC AUC-score på 0,86.

Contents

List of Figures

Chapter 1

Introduction

Every system has flaws. The security system we hide behind is only as good as the next hacker. It has become increasingly hard to hold the wall around our systems and keep intruders out and hence, we opt to move towards a detection paradigm instead of a perimeter defense. Looking at network traffic, can an anomaly detection algorithm effectively detect intrusions in unstructured time series data? This will be the question to answer in this report using a specific set of methods. There are many different approaches to the problem of intrusion detection but the goal here is, given raw data from the network traffic packets, to engineer features from the data that a machine learning algorithm can handle and output a decision of whether an attack occurred or not. This pipeline will be referred to as an anomaly detection model. In designing an intrusion detection system one must decide on one of the myriad methods available when it comes to network sniffing, host-based detection, data stream processing and machine learning algorithms. The problem at hand here is to detect an intruder in a system by analysing anomalies in the user behaviour. In order for this to be possible one must first preprocess the available network data in a way that makes it interpretable by a machine learning algorithm. That is to say, feature engineering from unstructured data is a key aspect to solving this issue. For feature engineering, the method of term frequency-inverse document frequency (tf-idf) is used. To detect anomalies, a Long Short-Term Memory (LSTM) Autoencoder is used. The data utilised throughout the project comes in the form of packet capture files from a red team-blue team exercise conducted in the Cyber Range And Training Environment (CRATE) at the Swedish Defense Research Agency (FOI). CRATE is a designated testing environment for conducting and logging cyber attacks. The red team-blue team penetration test consists of a so called blue team monitoring the network while a red team attempts to gain unauthorised access to it. Hence, the data from this activity contains network packets of normal behaviour as well as attacks. The anomaly detection models will be evaluated using the two metrics, the F1-score and the area under Receiver Operating Characteristic (ROC) curve. The benchmark available to compare time series anomaly detection against come from Numenta Anomaly detection Benchmark (NAB) [7]. These results are in the form of online anomaly detection of single value metrics. The application of NAB is different enough that the results cannot easily be compared to the situation in this project, therefore the more common evaluation metrics will be used so that they can be used for comparison in the future. If the intrusions from this dataset can be detected we would end up with an intrusion detection system worthy of notice. To reach this point we will first cover the background theory of existing anomaly detection methods, data processing software and security related software. Once all terms have been defined and understood, the technical approach will cover the specific methods and tools used in the proposed solution. This is then followed by a discussion on the results attained by testing the proposed solution.

1.1 Ethical Implications

Collecting network traffic for users on the network is a requirement for this project to provide accurate results. There are always going to be questions of ethics involved when logging user data, whether it is network traffic or other data. This is due to the fact that such data coupled with a machine learning algorithm can build up a profile of a users behaviour. This is done in many places already for the benefit of the user, from recommending movies to targeted marketing. It is important to keep in mind that collecting this data is not unethical, but deciding what to do with it needs to be carefully thought out. The data that is collected does not contain personal information and details, but is just the network packets during a normal day's work. Within this project, the data that is collected is network traffic from all users on a network and the sole purpose for storing this data is to train an anomaly detector to attempt to find intrusions on the network.

Chapter 2

Background

A great deal of research has gone into intrusion detection and anomaly detection. This section will highlight the main points of interest among these separate projects as well as existing methods.

2.1 Intrusion Detection

The National Institute of Standards and Technology classifies intrusion detection as "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network" [8]. The system that makes up an intrusion detector (IDS), consists of a number of components. A system to be monitored is the starting point. Within this system a set of sensors are deployed. What these sensors record varies across detectors, but is often raw network traffic. A database of known exploits or attacks can be implemented to aid the detector. These components will not be analysed deeply within this report, but the detector itself is the focus. The detector consists either of a single classifier or an ensemble of classifiers. This classifier then outputs a binary decision of whether an attack is happening or not. This information is sent to the response component and here the system can either simply alert or take measures to stop the attack.

Anomaly detection methods aim to gather data from users to create a profile which are defined as "normal" behavior. Commonly two profiles are maintained, a stored or reference profile and a current profile. The current profile is updated in real time as new data comes in and is successively compared to the reference profile to calculate any discrepancy. Advantages of such methods are that they can usually detect malicious

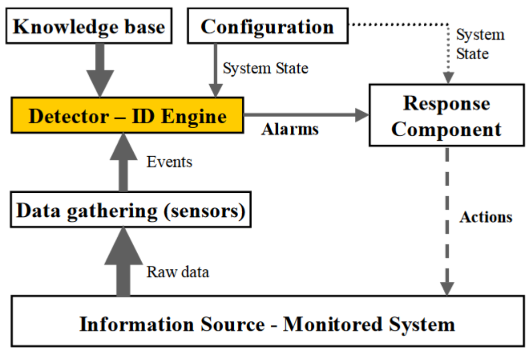


FIGURE 2.1: Basic architecture of an IDS from *"Intrusion Detection: A Survey"*[1]. Starting from the bottom of this figure, a system is monitored and raw data is gathered by sensors. This data is then processed and passed to the central component, the detector, here in yellow. This detector can have aid from a knowledge base of known attacks. The output of the detector is sent to the response component and an action is taken in the monitored system based on the output of the response component. This project will focus on the detector component and processing the raw data.

behavior which occur over extended periods of time. Some anomaly detection schemes opt to update the reference profile with incoming normal data, but one fatal drawback of this is that an attacker that is familiar with the detection system is theoretically able to slowly train it to accept malicious behavior as normal by submitting segments of malicious data that by itself appears normal but over time builds up into something malicious. The idea behind anomaly based intrusion detection stems from the assumption that malicious activity is a subset of anomalous activity. This type of intrusion detection system groups all anomalies as intrusions, which could result in a large number of false alarms. It does provide a way to recognize all anomalous behaviour though, and this allows for further analysis on a smaller set of events. A set of established methods to detect anomalous behaviour will be explained hereafter.

2.2 Bayesian Networks

Bayesian networks can be seen as probabilistic directed acyclic graphs (DAG) together with a set of conditional probability tables (CPT) which represents a set of random variables and their conditional dependencies. Due to the ability to represent causal relationships Bayesian networks can be used to predict consequences of actions. The joint probability function in general is the product of all variables conditioned on their direct parents in the DAG. Let there be n variables denoted $x_1, ..., x_n$ built as a DAG with n vertices V, then the joint probability is:

$$p(x_1, ..., x_n) = \prod_{i \in V} p(x_i | x_{parents(i)})$$

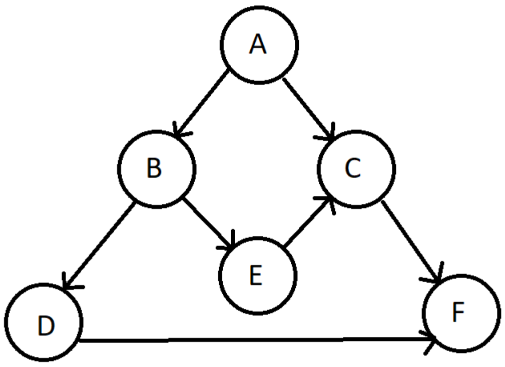


FIGURE 2.2: Directed Acyclic Graph showing causal relationships between variables as an example for a Bayesian Network.

As an example, Figure 2.2 shows a DAG with six vertices labeled $A - F$. The joint probability for this graph is:

$$p(A, B, C, D, E, F) = p(D|B)p(F|D, C)p(C|A, E)p(E|B)p(B|A)p(A)$$

Valdes et al. [9] used naive Bayesian networks to implement intrusion detection on bursts of traffic. The model considered temporally contiguous traffic taken from a specific IP address and used this as a session. The model is built up as a tree structure where each node is given a prior π from its parents and a likelihood λ from its children and each node can take on a number of discrete states. The relation between likelihood and prior is given by the conditional probability table

$$CPT_{ij} = P(state = j | parentState = i)$$

where

$$CPT_{ij} \geq 0, \forall i, j$$

$$\sum_j CPT_{ij} = 1, \forall j$$

Downward propagation of the prior to all children is done by:

$$\pi(node) = \alpha \pi(parentNode) \cdot CPT$$

as a left multiplication of the $\pi(parentNode)$ vector and the CPT matrix and α is a normalisation constant to keep the sum of the result at one. The upward propagation of likelihoods is done by:

$$\lambda^{parent}(node) = CPT \cdot \lambda(node)$$

$$L_i(parent) = \prod_{c \in children(parent)} \lambda_i^{parent}(c)$$

$$\lambda_i(parent) = \frac{L_i(parent)}{\sum_j L_j(parent)}$$

Where L represents the element-wise product

By considering traffic to a protected resource such as a server within the protected local network they are able to detect distributed attacks. Once an event is registered, all active sessions are inspected and if a match is found the matching session is updated, otherwise a new session is initialised. The session list might grow very large and therefore a scheduled "housecleaning" operation which deallocates sessions with timouts before the "housecleaning" was called. If the session list reaches maximum capacity and a new session is starting, the session with the most distant event time is deallocated. A disadvantage of this scheme is that if the session list is configured to be too small, it may reach maximum capacity quickly and start deallocating live sessions. This way, some true positives could be missed. An advantage of this model though is that it is able to detect attacks where the separate attacks would not raise alarms, given that the sessions are kept alive.

Another area where Bayesian networks have uses is in suppression of false alarms. Kruegel et al. [10] proposed an idea to replace the simple decision process based on thresholds with a Bayesian network in order to reduce the number of false alarms in an IDS. The issue that is addressed here is that with simple aggregation and threshold decision one may run into problems such that the threshold is too high for any single sensor to elicit an alarm or that it is too low and hence a surplus of alarms are signaled. They way [10] does this is by building a network consisting of a root node representing a two-state variable, normal and anomalous. Each model gives rise to a child node to capture the respective outputs from the models. In addition, a confidence node is introduced as to describe the weight that each model will have on the overall decision. This way the cases where values of a particular feature have a high variance will not have as great of an impact on the final decision. Using this implementation, [10] shows experimental results of lowering the false positive rate.

2.3 Principal Component Analysis

Recently in machine learning scenarios, we are required to handle large sets of multi-dimensional data. Dimensions are the number of random variables or features. With a growing feature space, the computation time can become infeasibly large, thus making use of appropriate dimensionality reduction can lead to great performance boosts. Principal component analysis (PCA) is a method to transform a number of possibly correlated variables into a set of linearly uncorrelated variables. Correlation between features describes the behaviour in which two or more features act in unison. A positive correlation would indicate that when one feature increases so does the other, a negative correlation implies that one grows and the other decreases, and orthogonal features are uncorrelated. The principal components are also referred to as eigenvectors. There exists exactly as many eigenvector and eigenvalue pairs as dimensions in the data. The eigenvector represents a direction along the data and the eigenvalue describes the variance along that vector. The principal component is another name for the eigenvector with the largest eigenvalue. To calculate the PCA we first need to compute the eigenvectors and eigenvalues of the covariance matrix. If we let X be the dataset under analysis and $E[X]$ the expected value of X, then the covariance matrix Σ is defined as follows, assuming X has zero mean:

$$\Sigma = E\Big[(X - E[X])(X - E[X])^T\Big]$$

Once this is computed, the eigenvalues are found by introducing a scalar λ.

$$A\vec{v} = \lambda\vec{v}$$

The eigenvalue problem is, solving the above equation where A is an $n \times n$ matrix, $\vec{v}$ is a non-zero $n \times 1$ matrix and λ is a scalar. We can rearrange this equation and use the fact that $\vec{v}$ is non-zero to arrive at this form:

$$A\vec{v} - \lambda\vec{v} = 0$$

$$(A - \lambda I)\vec{v} = 0$$

where I is the identity matrix. Since $\vec{v}$ is non-zero, the only solution to this equation is when

$$|A - \lambda I| = 0$$

The solution to this equation will be n scalars denoted $\lambda_1, ..., \lambda_n$. Each of these eigenvalues will correspond to an eigenvector which is computed by inserting each eigenvalue back into $(A - \lambda_i)\vec{v} = 0$.

$$\Sigma\vec{v}_i = \lambda\vec{v}_i$$

By doing these computations with $A = \Sigma$ we will get the principal components of our dataset X.

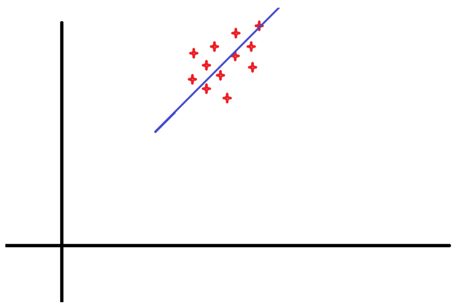


FIGURE 2.3: Visual representation of PCA using sample dataset and only the first principal component

Seen in the figure is the eigenvector with largest eigenvalue or direction with greatest variance. Once this is found, it is possible to reduce the dimensionality of the dataset by projecting the data to all eigevectors with an eigenvalue greater than some threshold. Shyu et al. [11] proposed an anomaly detection system where PCA looked for outliers as anomalous points. Instead of using the Mahalanobis or Euclidean distance as a measure of whether a point is an outlier, considering the principal components of a multivariate data set we can gather information about the nature of the outlier. Such information as whether the outlier is an extreme value or them having a different correlation structure to the normal instances. In order to achieve this both the major and minor components are taken into account. Minor components are those with eigenvalues below 0.2 which indicates that there is some relationship among the features. An observation is classified as an attack if either of the following two expression hold:

$$\sum_{i=1}^{q} \frac{y_i^2}{\lambda_i} > c_1$$

$$\sum_{i=p-r+1}^{p} \frac{y_i^2}{\lambda_i} > c_2$$

where p is the number of principal components, q the number of major components, y_i are the principal components of an observation, λ_i the corresponding eigenvalues, r are the number of minor components and c_1, c_2 are outlier thresholds. By using these two thresholds in combination, the scheme is able to provide information about the nature of the outlier. The first function checks if a point is extreme in an orignal feature whereas the second function detects points that do not conform to the normal correlation structure. This method to reduce the dimensionality of the audit data generated better detection rates than other outlier based anomaly detection algorithms. Using PCA as an anomaly detection scheme allows for real time detection and does not require assumptions to be made about the data. A drawback of this method is that there is no way to distinguish between different attacks, although it does allow for unsupervised anomaly detection.

2.4 Support Vector Machine

A support vector machine (SVM) is solution to the two-class classification problem by introducing a mapping to a high dimension and splitting the data by a maximum soft-margin hyperplane [12]. The maximum soft-margin hyperplane splits the two classes yet allows some misclassifications which allows for an optimal solution even in inseparable cases. Given $x_i \in \mathbb{R}^p, i = 1, ..., n$ as a training set and the vector $y \in -1, 1^p$, where $-1, 1$

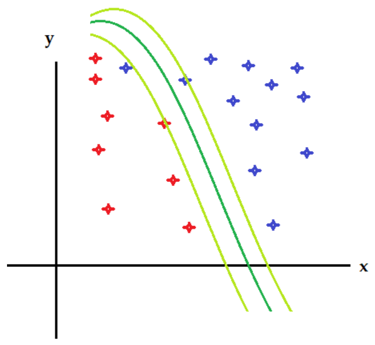


FIGURE 2.4: Example of two-class classification problem using an SVM with margin and two support vectors.

are the two classes. The general form of a hyperplane is:

$$w \cdot x + b = 0$$

where $\vec{w}$ is the weight vector, x is a training point and b is the bias term. Creating a boundary, we define the negative and positive classification boundaries to be, respectively:

$$w \cdot x_n + b = -1$$

$$w \cdot x_p + b = 1$$

where x_n and x_p are points belonging to the negative and positive class respectively. The distance between the two boundaries is then:

$$\text{distance} = \frac{2}{\|w\|}$$

The hard-margin problem can then be expressed as:

$$\min \|w\|, \text{ subject to } y_i(w \cdot x + b) \geq 1, \forall i$$

In order to extend the SVM to allow for a soft-margin hyperplane we introduce positive variables $\xi_i \geq 0$ and the new optimisation and constraint is:

$$\min \left(\|w\| + C \sum_i \xi_i \right)$$

$$y_i(w \cdot x + b) \geq 1 - \xi_i, \forall i$$

where C is a regularisation constant.

One of the advantages of an SVM is that when deciding which separating hyperplane to pick out of the infinite available options of separating hyperplanes, the SVM will pick the hyperplane with the largest margin to each class. Another reason that makes the SVM a strong machine learning model is the ability to use non-linear kernels which allow a separating hyperplane in an otherwise linearly inseparable case. Although, the problem with this is that this kernel needs to be picked beforehand and it can be difficult to decide which kernel suits the data best.

2.5 Markov Models

A Markov model of a system is a set of states of that system $S = s_1, s_2, ..., s_t$, the allowed transitions between those states and the respective probabilities of those transitions. What is important to note here is that each state is visible to an observer, this distinguishes this model type to a hidden Markov model which will be detailed shortly. At each time step in any given state, there is a probability to move to another state or to remain in the current state. This probability is conditioned solely on the current state. For example, a two-state system $S = s_1, s_2$ has moved between its states in a manner such as s_1, s_2, s_2, s_1. The probability to move to s_2 or stay in s_1 is independent on the first three time steps and depends only on the fact that it is currently in s_1.

$$p(s_t|s_{t-1}, s_{t-2}, .., s_1) = p(s_t|s_{t-1})$$

Ye et al. [13] presented an anomaly detection scheme based on Markov chains which

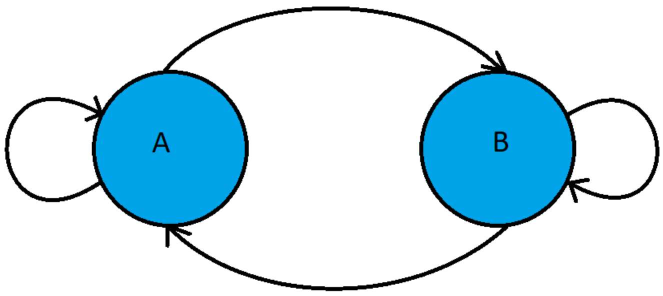


Figure 2.5: Markov model with two states showing possible transitions between A and B.

analysed system-call event sequences in Windows. A normality score was assigned to each state where the audit event was examined. A high such score would indicate that this is in fact normal behavior and a low score would point to suspicion of intrusion. Hidden Markov models (HMM) are an extension of the regular Markov models where the state sequence is not directly observable but needs to be inferred from the visible output observations. Let a set of observations $o = o_1, ..., o_t$ be the only observable output. From Bayes Theorem we have:

$$p(s_i|o_i) = \frac{p(o_i|s_i)p(s_i)}{p(o_i)}$$

and for a sequence of length t:

$$p(s_1, ..., s_t|o_1, ..., o_t) = \frac{p(o_1, ..., o_t|s_1, ..., s_t)p(s_1, ..., s_t)}{p(o_1, ..., o_t)}$$

The Markov assumption is that the probability of being in a state is only dependent on the state of the previous time step. Hence a sequence can be rewritten as:

$$p(s_1, ..., s_t) = \prod_{i=1}^{t} p(s_i|s_{i-1})$$

The other assumption that can be made is that each observation is independent of the other.

$$p(o_1, ..., o_t|s_1, ..., s_t) = \prod_{i=1}^{t} p(o_i|s_i)$$

Now the previous expression can be written:

$$p(s_1, ..., s_t|o_1, ..., o_t) = \prod_{i=1}^{t} p(o_i|s_i) \prod_{i=1}^{t} p(s_i|s_{i-1}) \frac{1}{p(o_1, ..., o_t)}$$

Given the observed output sequences, the task is to find a maximum likelihood estimate of the parameters of the model. A well known method is the Baum-Welch algorithm which is a special case of the Expectation-Maximization (EM) algorithm.

Let $\lambda = (A, B, \pi)$ where

$$A = A_{ij} = P(S_t = j|S_{t-1} = i)$$

$$B = B_j(o_t) = P(O_t = o_t|S_t = j)$$

$$\pi = \pi_i = P(S_1 = i)$$

The expression for A follows from the Markov assumption and describes the transition matrix for the states. The expression for B describes the probability of an observation o_t occurs in state at time t, where there are K possible observations. The last expression is defined as the initial state distribution and describes the probability of starting in each of the states. The HMM can be defined by λ with the three parameters. Now, given an observation sequence $O = (o_1, o_2, ..., o_t)$ the task is to find a set of parameters λ that maximises the probability that the observation sequence came from this specific HMM $P(O|\lambda)$. The Baum-Welch algorithm [14] finds a local maximum to this problem and is defined as follows:

- Initialise $\lambda = (A, B, \pi)$ with random parameters
- Let $\alpha_i(t) = P(O_1 = o_1, ..., O_t = o_t, S_t = i|\lambda)$
- - This is calculated recursively forward
  - $\alpha_i(1) = \pi_i B_i(o_i)$
  - $\alpha_i(t+1) = B_i(o_{t+1}) \sum_{j=1}^{N} \alpha_j(t)\alpha_{ji}$
- Let $\beta_i(t) = P(O_{t+1} = o_{t+1}, ..., O_T = o_T|S_t = i, \lambda)$ where $o_{t+1}, ..., o_T$ is the ending of the sequence.
- - This is found recursively, starting at the end.
  - $\beta_i(T) = 1$
  - $\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) A_{ji} B_j(o_{t+1})$
- A temporary variable is introduced

$$\gamma_i(t) = P(S_t = i|O, \lambda) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

This is the probability of being in state i at time t given the observation sequence O and HMM parameters λ.

- A second temporary variable is introduced

$$\xi_{ij}(t) = P(S_t = i, S_{t+1} = j|O, \lambda) = \frac{\alpha_j(t)A_{ij}\beta_j(t+1)B_j(o_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i(t)A_{ij}\beta_j(t+1)B_j(o_{t+1})}$$

This is the probability of being in state i at time t and state j at time $t+1$ given the observation sequence O and the parameters λ.

- The initial parameters $\lambda = (A, B, \pi)$ can now be updated to

$$\pi_i^* = \gamma_i(1)$$

$$A_{ij}^* = \frac{\sum_{t=1}^{T-1}\xi_{ij}(t)}{\sum_{t=1}^{T-1}\gamma_i(t)}$$

$$B_i(k)^* = \frac{\sum_{t=1}^{T}\gamma_i(t)\delta_{o_t,k}}{\sum_{t=1}^{T}\gamma_i(t)}$$

where $\delta_{o_t,k}$ is the kronecker delta.

Warrender et al. [15] conducted a comparison of four methods to represent normal behavior and recognizing intrusion in system call datasets, namely simple enumeration of observed sequences, comparison of relative frequencies of different sequences, a rule induction technique and hidden Markov models. To acquire the parameters of the model such that it will be able to recognize normal behavior, sequences of normal events are used as training data and the Baum-Welch algorithm was used to train the model. The state transition model was a fully connected one and hence leads to high computation costs during training. It was noted that initialising the HMM with random weights lead to poor detection rates but once initialised with a pre-determined write-read state loop accuracy increased. It is true in all cases for HMMs that a good initialisation leads to faster training, but finding such a configuration is a task in itself. Depending so heavily on a good starting point is a flaw in this scheme. While HMMs are able to capture long sequences they fall short in computation time and complexity.

2.6 Genetic Algorithms

A genetic algorithm is a search heuristic that takes its ideas from natural selection in order to solve an optimisation problem. It is an iterative algorithm where the initial condition is having a random sample space of solutions, called population. A fitness function calculates how good a chromosome is and each candidate is a solution to the optimisation problem that has a chromosome-like data structure which can evolve by selection, mutation or crossover operators. A large number of random initial chromosomes are created if one chromosome looks like the following:

$$[A, B, C, D, E, F, G, H]$$

then a crossover operation is creating a new chromosome from two previous ones:

$$[A, B, (C, D, E), F, G, H]$$

$$[(I, J), K, L, M, (N, O, P)]$$

$$\Rightarrow [I, J, C, D, E, N, O, P]$$

a mutation operation may look like:

$$[A, B, C, (D, E, F, G), H]$$

$$\Rightarrow [A, B, C, G, F, D, E, H]$$

and selection would choose the chromosomes with highest fitness value and allow them to move into the next generation.

What emerges is an ever evolving system which strives to optimize itself to match the predefined heuristic. In intrusion detection, this algorithm translates to constructing chromosomes which encode the features as information and using a rule set to classify it. In the case of network traffic, a chromosome can be built up like the following example: The collected traffic information could consist of the following fields

- Source IP: 241.245.26.44
- Destination IP: 100.90.122.7
- Network Protocol: TCP
- Source Port Nr: 42535
- Destination Port Nr: 80
- Duration of connection: 120

Categorical fields can be converted to integers describing the index in a list that it is found in. We have a list of protocols [UDP,SSH,TCP] and hence TCP corresponds to 2.

If we separate the numbers in each IP address and append every value to an array we get the following:

$$[241, 245, 26, 44, 100, 90, 122, 7, 2, 42535, 80, 120]$$

This is now a chromosome built up from the information in the example.

Advantages of genetic algorithms are that they are flexible and robust as they work as global search methods. Moreover, genetic algorithms converges from multiple directions and is probabilistic by nature rather than deterministic. The first to try to implement genetic algorithms into IDS was Crosbie et al [16] by applying multiple agents to detect network based anomalies. The clear advantage of the method was that it used many agents to monitor a range of network based parameters, though there was no clear communication between the agents themselves. The genetic algorithm built by Crosbie et al used the following features:

- Total number of socket connections
- Average time between socket connections
- Minimum time between socket connections
- Maximum time between socket connections
- Destination port to which the socket connection is intended
- Source port from which the connection originated

Using these simple features, the genetic algorithm was able to detect attacks such as

- port flooding (rapid connections to specific port)
- port-walking (searching port space for vulnerable services)
- probing (gaining information from services)
- password cracking (rapid remote login attempts)

The fitness function used to train this model is described by the follwing expressions.

$$\delta = |outcome - suspicion|$$

The absolute difference between the agents suspicion level and the real outcome. This way, if the agent reports a detection but does not report a high level of suspicion after seeing the data, it will receive a low fitness score. Another metric was introduced, the penalty score, to deter the agents from misclassifying obvious attacks.

$$penalty = \frac{\delta \cdot ranking}{100}$$

where the ranking is a pre-defined variable for how difficult it is to detect each and every attack. The fitness is thus computed

$$fitness = ((100 - \delta) - penalty$$

While this simple model describes that genetic programming can be used for intrusion detection, it is lacking in coverage. Only very basic attacks can be detected.

2.7 Artificial Neural Networks

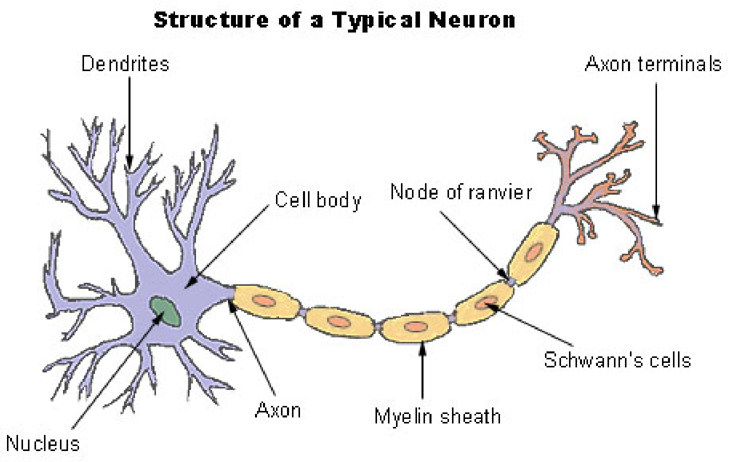


Figure 2.6: Depiction of biological neuron [2]

The idea of artificial neural networks is based on the biology of neurons in the brain as seen in Figure 2.6. Reconstructing this model in a computer hopes to replicate the abilities of the brain as well. Comparing this single neuron to the network of artificial neurons below in Figure 2.7 shows how the parts of the biological neuron remains. The dendrites are inputs, the axons outputs and the cell body is the weight with activation function. The machine version consists of a system of interconnected neurons that exchange messages with each other. These connections are defined by numeric weights that can be altered based on experience. A system such as this can be trained to recognize arbitrary data as input and map it to an output, which in the case of intrusion detection can be a binary descriptor. Even with a high tolerance to imprecise data and

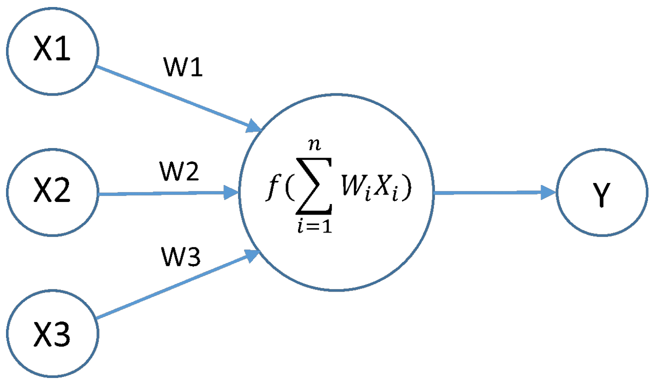


FIGURE 2.7: Activation of incoming signals in artificial neuron[3].

an ability to infer solutions without prior knowledge, ANNs can usually lead to heavy computation costs as the neural network needs to manipulate all the weights of the nodes.

Ghosh et al. [17] made use of feed-forward back propagation and the Elman recurrent network which showed better experimental results compared to the standard multi-layer perceptron based network. The Elman recurrent network is based on the feed-forward network but in addition include a set of context nodes. Each of these context nodes receive input from one hidden node and outputs it to all hidden nodes in the same layer. This recurrent fashion allows the network to retain information between inputs. The measure of anomaly of each sequence event is the difference between the output at time n and the input at time n+1. This can also be viewed as the error of predicting the next input given an output. The Elman network is described as follows:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

where h is a hidden layer, x is an input vector, y is the output vector, W is the weight matrix, U is the context matrix, b is the bias, and σ is an activation function. Recurrent neural networks have shown to perform well on intrusion detection cases due to their ability to take sequences into account.

Activation functions:

$$\text{tanh(x)} = \frac{2}{1+e^{-2x}} - 1$$

$$\text{sigmoid(x)} = \frac{1}{1+e^{-x}}$$

$$\text{hard sigmoid(x)} = \max(\min(u_s(x), 1), 0)$$

where $u_s(x)$ is a clipped Maclaurin expansion of the sigmoid function.

$$\text{sigmoid} = \sum_{n=0}^{\infty} \frac{(-1)^{n+1}(2^{n+1}-1)B_{n+1}}{n+1} x^n$$

where B_n is the Bernoulli number which can be defined as

$$B_n = \frac{n!}{2\pi i} \oint \frac{z}{e^z - 1} \frac{dz}{z^{n+1}}$$

which leads us to:

$$\text{sigmoid(x)} \approx u_s(x) = \frac{1}{2} + \frac{x}{4}$$

A common neural network is the feed-forward network where the information moves in only one direction, "forward" and no recurrence is allowed. What this in essence consists of is a set of weight matrices and differentiable activation functions. A network like this is trained using back-propagation of errors in combination with an optimization technique such as gradient descent. The training happens in two phases, the forward feeding of the input and the backward propagation of the errors to update the weights. Let the input feature vector be $X = \{x_1, x_2, ..., x_n\}$ with n instances each with d features. In the first layer of the network, the input vector X is multiplied by the first weight matrix w_1 and run through an activation function, for example:

$$\text{out} = \tanh{(X \cdot w_1)}$$

The result of this is sent on the the next layer and this is repeated until reaching the output nodes. The back-propagation is done by calculating the partial derivative of the error with respect to the weight

$$\frac{\partial E}{\partial w_{ij}}$$

which is solvable by use of the chain rule and the fact that the activation function is differentiable. Let *net* denote the net input in a cell from all its sources. Let *out* denote the output from a cell after *net* has been squashed by an activation function. Then the change in error depending on a particular weight w can be calculated as follows:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Only one term in *net* depends on w_{ij} and hence the others fall out.

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} w_{ij} out_i = out_i$$

The second term:

$$\frac{\partial out_j}{\partial net_j} = \frac{\partial}{\partial net_j} f(net_j)$$

where f is some differentiable activation function. If we assume to use the squared error function $E = \frac{1}{2}(t-y)^2$ where t is the target output and y is the output from the neuron. When looking at the output neuron, we have that $out_j = y$

$$\frac{\partial E}{\partial out_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2 = y - t$$

The weights are updated by adding

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

to the initial w and where η is a constant called the learning rate. [18]

2.8 Replicator Neural Network

The replicator neural network (RNN) is a variant of a feed-forward neural network where the input and output layers have the same size but the hidden layers have smaller sizes. Setting the target values as the input this forms an auto-encoder. The model is required to learn a compressed version of the input in order to reproduce the input at the output layer. The underlying idea here in terms of anomaly detection is that the model will be trained solely on so-called ”normal” or ”safe” data” (meaning regular behaviour on network or host). When unseen ”normal” data is introduced to the network, the mean square error between the produced output and the input is expected to be low. When unseen ”abnormal” data is introduced, the error is expected to be much higher and this is how the classification is done [19]. This method will be explored on new data in this project. The replicator neural network described in [19] uses three hidden layers of varying sizes between (30,3,30) to (45,3,45). Symmetry is an important factor in this type of model as otherwise the encoding and decoding stages can cause different calculations that are not due to an inability to model the training data but because they are containing a different number of weights. Hawkings et al. keeps the middle layer small at only three nodes which may be too small for some complex data. The whole model is expected to be represented by only three hidden nodes, which limits the complexity of the model itself. The activation function of the middle layer is a staircase function which aims to classify the different attacks. This is introduced in order to distinguish the different attacks but leads to larger room for error as the model for a whole attack needs to be within a small range of the staircase function. The staircase

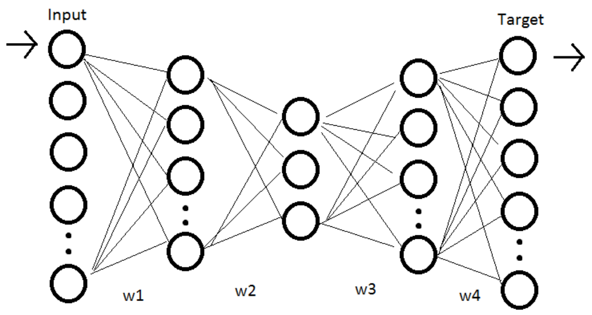


FIGURE 2.8: RNN with three hidden layers. This model requires the number of input neurons to be equal to the number of output neurons. For training, the target is set to be equal to the input and the network will replicate seen data and be unable to replicate any unseen data. This way anomalous data will yield a high reconstruction error (mean squared error between input and output)

function is defined as follows:

$$S_3(\theta) = \frac{1}{2} + \frac{1}{2(k-1)} \sum_{j=1}^{N-1} \tanh[a_3(\theta - \frac{j}{N})]$$

where a_3 is a parameter for the third layer and used as $a_3 = 100$ in the paper, N is the number of steps and set to $N = 4$ and k is index of the layer.
Without using such a staircase function, the classifier becomes binary and is only able to classify normal/anomaly but not which type. An important note for training a network like this one is that there should be a low number of anomalies in the training set or none at all, otherwise the model will learn to allow anomalies. This assumption is not a certainty depending on how the data was collected and can cause issues. Another point for this type of network is that it is heavily reliant on a threshold to determine what level of replication error shall be allowed. This threshold can be decided after looking at the results of the test data, but then the classifier may not perform well on novel data that is not in the training or test set.

2.9 Isolation Forests

The isolation forest technique is based on the assumption that anomalies are few in numbers and inherently different from the normal instances. Given a data set $X = \{x_1, x_2, ..., x_n\}$ with n data points, each with d features. The algorithm to find the number of partitions needed to isolate a chosen data point $x_{iso} \in X$ is shown below. This is a recursive algorithm, hence when X is used below, it denotes the current input set into the algorithm. Initially, this is the whole dataset, in subsequent steps it will be a subset of the full dataset. [20]

The algorithm with a two dimensional example for clarity. Given a dataset $X = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ we want to isolate (x_1, y_1) the first data point in the set.

1. Uniformly randomly select a feature between 0 and 1. This represents a selection between the x and y features, and in this example let us assume that y was selected.
2. Uniformly randomly select a value η between the minimum and maximum values of y found in X. The y values range from for example -5 to 2 so η is chosen in this range as $\eta = 0$.
3. Partition X on η. Two new sets are created, one with all points $(x_i, y_i) : y_i \leq 0$ and one with $(x_i, y_i) : y_i > 0$
4. Check which side contains the point (x_1, y_1)
5. Recursively repeat steps 1-5 until (x_1, y_1) is the only data point in a partition. It is then isolated.

This algorithm then generalises to the following:

1. Uniformly randomly select a feature ϕ from 0 to d.
2. Uniformly randomly select a value η between $\min(\{x_1^\phi, x_2^\phi, ..., x_n^\phi\})$ and $\max(\{x_1^\phi, x_2^\phi, ..., x_n^\phi\})$ where x_i^ϕ represents the i-th data point and feature ϕ.
3. Partition X on η. A partition on η is a single split of X such that one side contains data points $x_i : x_i^\phi \leq \eta$ and the other contains data points $x_i : x_i^\phi > \eta$
4. Check which partition contains the data point x_{iso}
5. Recursively repeat steps 1-5 on the partition containing x_{iso} until it is isolated or a maximum depth has been reached. It is considered to be isolated when it is the only data point in a partition.

Running this algorithm on all points will yield a tree structure where the root node represents the full dataset X and each branch represents a partition. The leaf nodes will all correspond to a single data point and the depth to this leaf node represents the number of partitions required to isolate the data point. This model is illustrated in the Figure 2.9.

Figure 2.10 illustrate the algorithm and how an anomalous data point is expected to be isolated quickly, whereas a normal data point is expected to need more partitions.

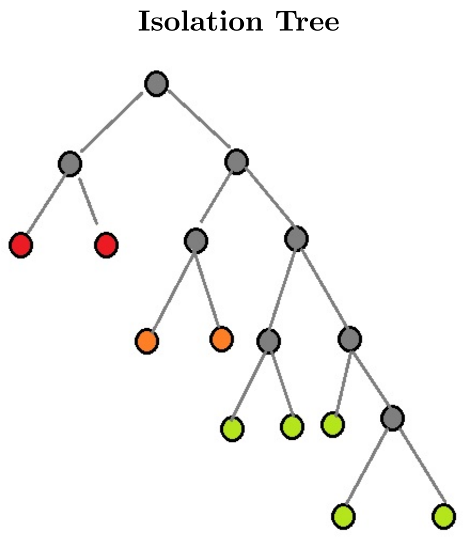


FIGURE 2.9: Leaf nodes with low depth are considered anomalies, and in this figure colored red. Leaf nodes with greater depth are considered normal and colored green here. The branches describe the partitioning of features. Many trees like this one are created and the average depth to each point is taken to be the deciding factor as to what is in fact an anomaly or normal

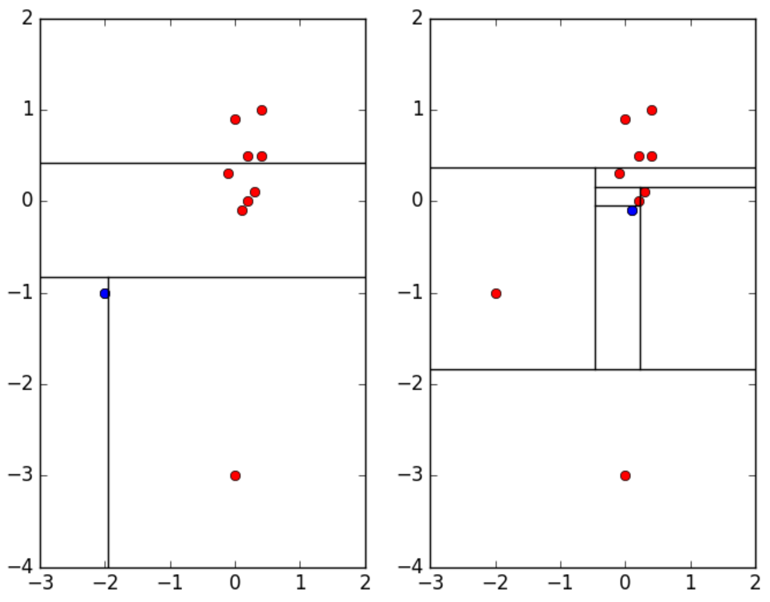


FIGURE 2.10: Figure on the left shows isolation of an anomaly, requiring only three partitions. On the right, isolation of a normal point requires six partitions

As opposed to other methods such as clustering or classification, isolation forests do not learn a profile for what is normal but directly attacks the anomalies. No distance metric is used in this algorithm which saves time on computations. Isolation forests have a linear time complexity. Figure 2.11 shows the difference between the average number of partitions required to isolate an anomalous point and a normal point.

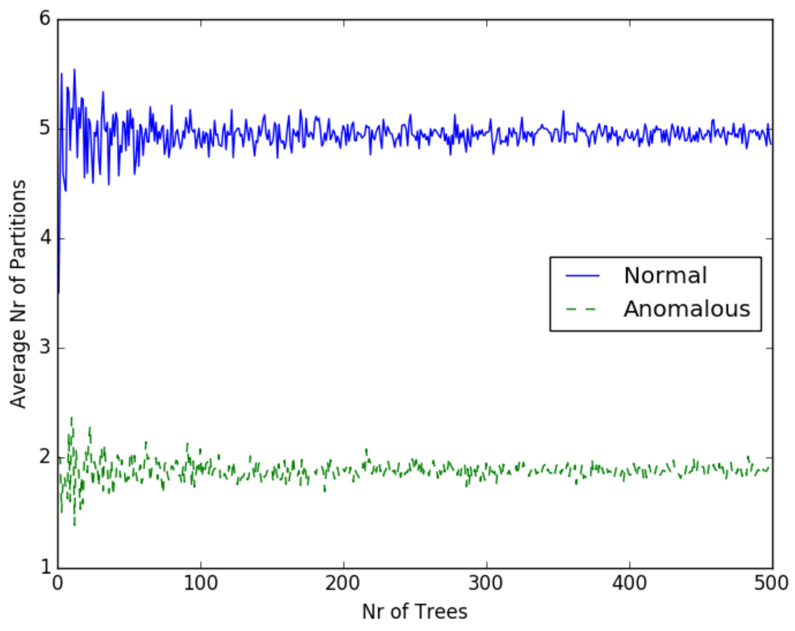


FIGURE 2.11: Figure showing the average number of partitions needed to isolate an anomalous point (blue) and a normal point (green) as the number of trees increase. The two data points are taken from the synthetic dataset shown in Figure 2.10

Figure 2.11 demonstrates the fact that the average number of partitions converges as the number of trees grows. A forest in this case simply represents the mean of all trees. The mean depth to each leaf node from all trees is output as the isolation forest classifier. At this point a Receiver Operating Characteristic (ROC) curve can be created with a moving threshold between the minimum and maximum depths of leaf nodes.

The greatest advantage of isolation forests is the computational speed compared to other methods. It is a simple model and hence does not require long training times, the complexity grows linearly with samples and features. At the same time, it is a simple model which may lead to misclassifications of complex data. Since there is no consideration taken to time, an isolation forest will fall short in detecting intrusions that span a greater temporal range. To solve this problem, the following methods are considered.

2.10 Long Short-Term Memory

A long short-term memory network is a recurrent artificial neural network with some extensions. The idea of a recurrent neural network is that not only is information only provided through the input layer, the hidden layer of a previous iteration can now pass information to future iterations, as demonstrated in Figure 2.12. How the information

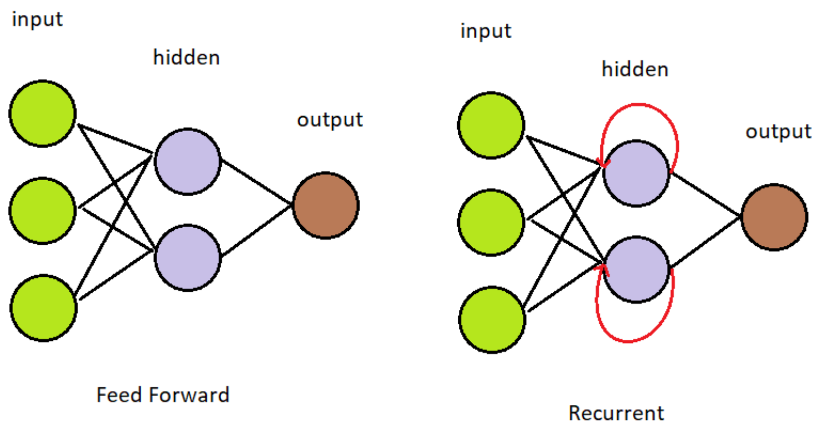


Figure 2.12: Main difference between feed-forward network and a recurrent network. The output of the hidden layer is fed as input into the next iterations hidden layer.

is fed back into the network will change the structure of the network but the general idea of recurrent networks is just to feed information back into itself. The immediate reaction to considering such a network is the idea that sequences and series are now taken into account. An element of time has been introduced, which was not present in the feed-forward network. Such networks perform well on for example speech recognition tasks.

A very special version of a recurrent neural network is the Long Short-term Memory network or LSTM deviced by Hochreiter and Schmidhuber in 1997 [4]. Regular recurrent networks perform well when the gap between two connected pieces of information is not very large, but quickly run into trouble when trying to relate one event to another event occurring many steps earlier. That is to say, the error signal when propagated through time tends to either blow up or vanish. In order to solve this problem and allow for constant error flow, the LSTM introduces gate units to regulate the flow of information and error. The two points of interest here are the gates. One multiplicative input gate unit to protect memory contents from irrelevant input and one multiplicative output gate unit protects other units from currently irrelevant stored memory. The activation of in_j at time t is $y^{in_j}(t)$ and similarly, the activation of out_j is $y^{out_j}(t)$ which are defined as follows:

$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t))$$

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

where

$$net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1)$$

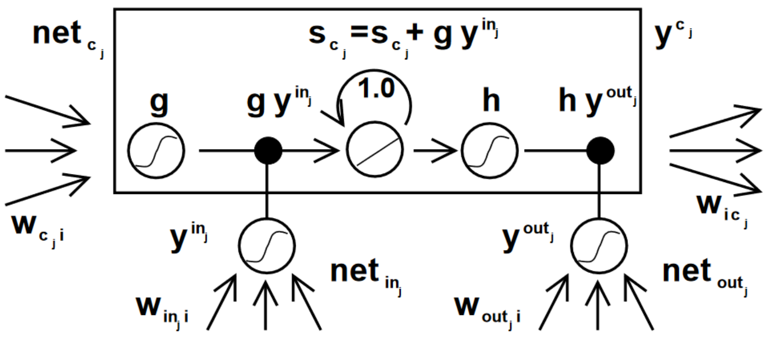


FIGURE 2.13: Memory cell taken from the LSTM paper[4] describing the architecture of the gate units. The differentiable function g squashes the input from net_{c_j} and the differentiable function h scales memory cell outputs from the internal state s_{c_j}

$$net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1)$$

and

$$net_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1)$$

The summation indices u can stand for input units, gate units or hidden units. The differentiable activation function f squashes the current net input $net(t)$ and w_{ij} is the weight on the connection from unit j to i. The output from memory cell j, c_j at time t is

$$y^{c_j}(t) = y^{out_j}(t)h(s_{c_j}(t))$$

the internal state $s_{c_j}(t)$ is

$$s_{c_j}(0) = 0$$

$$s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t)g(net_{c_j}(t)) \text{ for } t > 0$$

The gates allow for the network to use in_j to make decision regarding keeping or overriding information in a cell, c_j and out_j to decide when to access cell c_j. The error flow needs to be learned by the network and controlled by the input and output gates. When done correctly, the error signal will not blow up or vanish, leaving a system able to learn long sequences and distant causes and effects. Not much work has been done on intrusion detection of sequential data using LSTMs on unstructured data and this will be explored further in this project. The proposed idea of using LSTMs as an intrusion detection tool is to build an autoencoder structure, much like the Replicator Neural Network, and apply it to sequential data. The input to the network will be a tensor of the shape [batch size, sequence length, features] and the output will be the same tensor. The proposed solution will be able to replicate sequences of feature vectors. The training data will be only normal data and the test data will contain intrusions. Once the model has learned what normal sequences are, it will produce a low error of recreating the input. Anomalous data, which is inherently different from normal data will lead to a larger reconstruction error and hence sound an alarm.

2.11 Neural Turing Machine

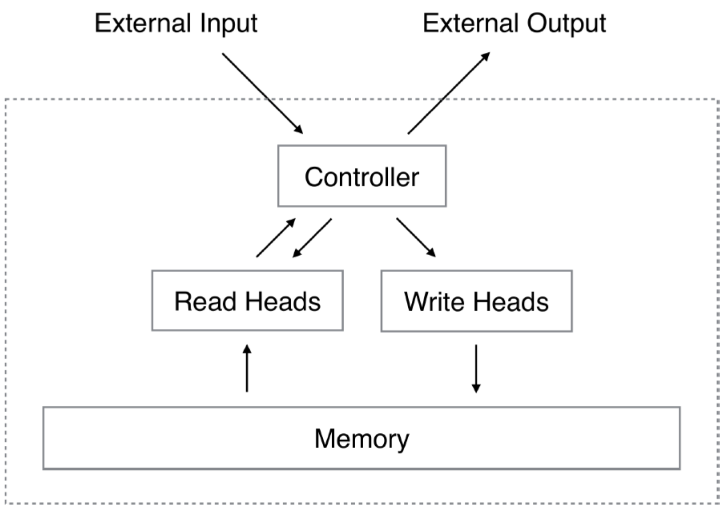


FIGURE 2.14: High-level architecture of Neural Turing Machine where the dashed box is the Neural Turing Machine. The controller is the interface between the external world and the internal system. The controller also interacts with a memory matrix by read and write operations. [5]

The Neural Turing Machine (NTM) consists of two main parts, a neural network controller and a memory bank. What differentiates this system architecture to a regular neural network is the ability to interact with the memory matrix using read and write operations. Although, like a regular network, each component of the architecture is differentiable which makes it simply trained by gradient descent. This is done by allowing a "blurry" memory access, where a normalised weighting is defined over the memory locations which dictates the degree to which a head will read or write this memory location.

If the memory matrix is of size $N \times M$ where N is the number of memory locations, M is the length of each memory vector and $\mathbf{M}_t(i)$ is the contents at time t of memory row i. The N elements $w_t(i)$ of the weighting matrix $\mathbf{w}_t$ follows the normalistaion constraint:

$$\sum_i w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1, \forall i$$

Then the read vector $\mathbf{r}_t$ of length M returned by the head is defined as:

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i)$$

The writing mechanism draws inspiration from the gates of LSTM. One write operation consist of an *erase* followed by an **add**. Provided the weighting $\mathbf{w}_t$ from the write head at time t, together with an *erase* vector $\mathbf{e}_t$ with M elements all in the range (0,1), the modification to the

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)[\mathbf{1} - w_t(i)\mathbf{e}_t]$$

where $\mathbf{1}$ is a row-vector of ones, and the multiplication is point-wise to the memory locations. A write head also creates an *add* vector $\mathbf{a}_t$ of length M which is applied after the erase step.

$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\mathbf{a}_t$$

The combination of erase and add operations from all write heads produces the content of the memory at time t. Both erase and add operations are differentiable, meaning that the composite write operation also is. Results from [5] indicate that NTM perform even better than LSTMs on long sequences. The implementation of NTM for intrusion detection will be left for future work and is an interesting area of study.

2.12 Data processing software

For any big data analysis, the use of data stream frameworks is required. There is a great variety of options and some of the most common ones will be mentioned here.

2.12.1 Apache Spark

Apache Spark is an open source cluster computing software that was developed to fill the gaps of the MapReduce cluster computing paradigm. Spark is centered around a data structure called resilient distributed dataset. The stream component of Spark utilises mini-batches to mimic a stream processing behaviour but comes at a cost of latency equal to the length of the batch.

2.12.2 Apache Flink

Apache Flink is an open source platform and a "stream dataflow engine providing data distribution, communication and fault tolerance for distributed computations over data streams.” Flink can process data as batches or as a stream. Flink only supports Java and Scala which is somewhat limiting.

2.12.3 Esper

Esper is an open source event series analysis and event correlation engine and aims to answer the question ”how to make sense of all the events flowing through your system?”

2.12.4 Apache Storm

Apache Storm is a stateless real-time stream processing software software aimed at analyzing big data. It follows a Master/Slave architecture similar to Hadoop with a Zookeeper based coordination. The greatest benefit of choosing Storm is that it is open source and very user friendly and can be used on a small or large scale. With respect to this project, as little time as possible should be spent on learning to use the related software and hence this is a great advantage of Apache Storm. Another benefit of choosing Storm over any other software is that it can support any programming language. This relates back to the previous point that less time should be spent on surrounding problems such as working with a language one is not comfortable with.
The use of this software in the scheme of the project will be to process all the stream of incoming data traffic and through analysis make it easier to handle. The output will feed right into one or more machine learning algorithms for detection.

2.13 Security Software

In working with security data, a number of different software are used to collect and manage data such as network traffic and host based logs. A number of software used in this project along with alternatives are mentioned here.

2.13.1 CRATE

CRATE is a Cyber Range And Training Environment at FOI consisting of 800 servers running virtual machines for intrusion testing. Within CRATE are some predefined systems such as industrial control systems, social network services and search engines. Yet, CRATE allows for the freedom to configure manual systems according to project specific specifications. It is in CRATE that one of the datasets used in this project was collected.

2.13.2 OSSIM

The Open Source Security Information Management system is a tool aimed at helping network administrators with intrusion detection and prevention. By combining log and risk management OSSIM gives the network administrator a great overview of all the security related features of a system. Multiple components work together to make OSSIM what it is, these include OSSEC which is a host-based intrusion detection system (HIDS) and Snort which is a network-based intrusion detection system (NIDS). OSSIM will act as the overhead manager of the whole system in this project. All other sensors will report to OSSIM so that everything can be viewed and managed from one location. This manager will only be installed on a single server in the network of virtual machines.

2.13.3 OSSEC

OSSEC is a scalable, multi-platform open source Host-based Intrusion Detection System (HIDS) integrating log analysis, rootkit detection, file integrity checking, Windows registry monitoring and real-time alerts. This software will take care of all the host-side activity and log any suspicious behaviour to OSSIM. Having this software separate from the network traffic will make things easier to handle by being able to focus on a smaller set of data, i.e. host-side events. OSSEC will not be installed on all of the machines in the virtual network but only a handful of so called vantage points. These points will be where surveillance is deemed important. OSSEC has built-in threat analysis and will only submit an alert when a threat threshold is reached. This greatly reduces data incoming to the machine learning algorithm.

2.13.4 Tripwire

Tripwire competes with OSSEC as an HIDS and comes in two version, an open source and a full-fledged enterprise version. While the open source version does contain all of the bare-bone features of a solid HIDS, to gain access to features such as multi-platform support and centralised control one needs the enterprise version. This project will be using OSSEC since it is only comparable to Tripwire Enterprise which comes at a cost and is infeasible for this project.

2.13.5 Snort

Snort is an open source NIDS able to perform real-time traffic analysis and packet logging on IP networks. Out of the three available modes, sniffing, logging and intrusion

detection, the last is the most versatile as it will monitor network traffic, compare it to a user-defined set of rules and take a predetermined action as a response. Snort can be used to detect attacks like buffer overflows, stealth port scans, OS fingerprinting and server message block (SMB) probes.
Working in tandem with OSSEC, this tool will make up the network-side defense of the system. Snort and OSSEC together will cover both host and network based traffic and they will relay their data to OSSIM to act as a manager. Snort will be set out to monitor the traffic on a few decided points of the virtual network and report back to OSSIM.

2.13.6 Suricata

Suricata is an alternative to Snort as a Network-based Intrusion Detection System (NIDS) with the main difference of having the capability of multi-threading, which Snort lacks. Suricata is a newcomer in the field of intrusion detection compared to Snort and therefore has much less support and documentation. While these two pieces of software are fair competitors, Snort will be used for this project for the reason being that it is much more well established.

2.13.7 SVED

This Scanning, Vulnerabilities, Exploits and Detection tool allows for repeatable network security tests on a given network. It will be used to test the network and host side security of the virtual ICS system. SVED includes features to design, execute and log exploits as well as submitting alerts from intrusion detection systems. The log has the following structure: "status, log_source_id, time_stamp, log_source_type, data, id". Aiming to substitute the expensive alternative of using red teams to compromise a system, SVED offers low-effort manual attacks to be performed in a precise sequence. Using this tool, recording malicious experiments becomes greatly simplified and testing specific attacks at specific points of the network will become reliable. This tool can also be used to generate background data or normal usage data in order to have something to compare the intrusions to. This is done by sending user behaviour to a bot on the network using a preexisting python script.

Chapter 3

Technical Approach

The proposed solution to the previously stated problem will be covered below. First a set of algorithms must be chosen, data must be generated, this data needs processing and finally features need to be selected. The data that will be regarded in this report will be Transmission Control Protocol (TCP) dumps from the DARPA KDD 99 dataset provided by MIT Lincoln Labs as well as event log file provided by FOI. The two different datasets are of different forms, the DARPA set is a list of TCP dump data and can be considered a classification task. The FOI log file is a time series analysis problem. For the DARPA set, Isolation Forest and Replicator Neural Network will be compared. For the time series analysis dataset from FOI, an LSTM Autoencoder will be used.

3.1 DARPA Dataset

The DARPA KDD 99 dataset is a network intrusion detection dataset used in The Third International Knowledge Discovery and Data Mining Tools Competition [21]. The dataset uses a version of the previous 1998 dataset constructed by MIT Lincoln Labs. Nine weeks worth of raw TCP dump was acquired from a simulated U.S. Air Force LAN. This data is labelled and has four categories: DOS, R2L, U2L, and probing. The full DARPA set contains 38 numerical features and about five million data points where around half was labelled as normal and half abnormal. Three of the features are categorical and the rest are all numerical.

3.1.1 Data Processing

The data from DARPA is already structured. The comma separated value file (CSV) can easily be converted into a two-dimensional NumPy array containing mixed categorical and continuous data. This was done by a short python script and the array is kept in memory during training. The categorical data was mapped to integers. The training time of neural networks can be reduced by scaling the data to a range of (0,1). In this case, the min-max scaling method was used to scale each feature such that the maximum value is mapped to 1 and the min value is mapped to 0.

$$X_{scaled} = \frac{(X - \min X)}{(\max X - \min X)}$$

3.2 FOI 2011 Dataset

The dataset from FOI recorded in 2011 was a red team/blue team intrusion detection test run in CRATE. The dataset consists of raw network traffic for three days of testing along with notes from the two teams of time stamps and descriptions of attacks initiated and detected. Figure 3.1 depicts the architecture of the network that was used in the collection of the data used in this project.

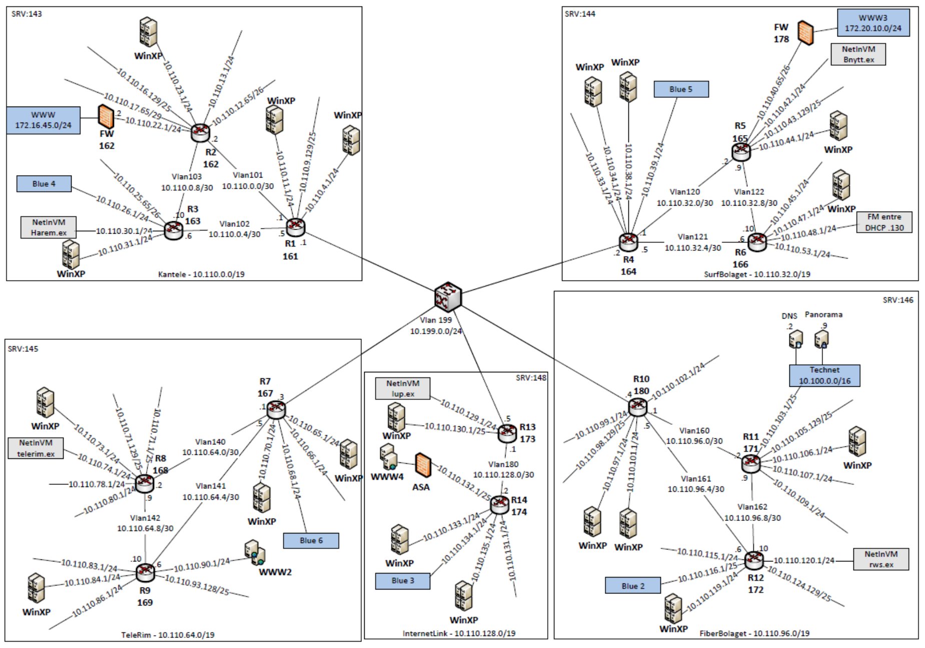


FIGURE 3.1: Infrastructure of system during FOI 2011 data collection [6]. System consisting of numerous Windows XP machines, switches and servers. Five blue team sensors are shown here where the system is monitored.

3.2.1 Data Processing

The data acquired from the penetration test at FOI requires much more processing than the already structured DARPA set. The first step in extracting features from the FOI dataset was to read from the raw network traffic file and organise the information into a structured array. The source and destination IP addresses were found in the data and converted to categorical data by indexing. Each IP address was mapped to an integer between 1 and the number of IP addresses found. The same was done for the protocol used for the packet. The last bit of structured data is the connection length which was saved as an integer and counted in milliseconds. No other data could be easily extracted and the rest was up to analysing the term frequency of the log entries against the whole log. The term frequency is in its most basic form just the raw count of each term in a document denoted $f_{t,d}$

$$tf(t,d) = f_{t,d}$$

The document frequency is defined as the number of documents d in a corpus D containing a term t.

$$df_{t,d} = |\{d \in D : t \in d\}|$$

and the inverse document frequency is the log scaled inverse fraction

$$idf(t,d) = \log \frac{|D|}{|\{d \in D : t \in d\}| + 1}$$

the plus 1 to account for cases where the term is not in the corpus. Finally, the tf-idf is the product of these two.

$$tfidf(t,d) = f_{t,d} \log \frac{|D|}{|\{d \in D : t \in d\}| + 1}$$

Tf-idf is a statistical measure of the importance of certain terms in a corpus but can also be used to provide a numeric representation of unstructured data. This makes it ever so useful for machine learning techniques, as some numeric data representation is necessary. Within this project, this method will be applied to the log files to produce a sparse matrix which will be used for training.

Once this two dimensional matrix was appended and all data was stored, it was then scaled by a min-max scaler to squash everything to a range of 0 to 1.

3.3 Machine Learning Algorithms

The three chosen machine learning algorithms for this purpose are the Replicator Neural Networks (RNN) , the Isolation Forests and Long Short-Term Memory (LSTM). The former two methods have shown high potential when dealing with network security dummy data in the form of TCP dumps. The algorithms are chosen for their respective strengths. Deep RNNs have the ability to model very complex behaviour but take some more computational effort. Isolation Forests are very computationally efficient and do a good job of locating outliers, yet they have a tendency towards shaping the region of inliers along the axis of the split.

3.3.1 Isolation Forest

Given the DARPA set the Isolation Forest technique counts the number of required splits to make in order to isolate each individual data point in 38 dimensional space. This is a semi-supervised classification problem since in the training stage, only "normal" data is provided and the algorithm compares the path lengths of the test data to the average of all the input training data. Given that all the training data is normal, we expect the average of all paths to be relatively high. When new test data is introduced containing anomalies, these are expected to deviate from the normal set and hence have a very low isolation path. The assumption is that most of the training data will be similar to each other, this then leads to many random splits required to isolate each data point. When an anomaly, which is assumed to be dissimilar to the training data, is presented to the algorithm, it should only take a few random splits to isolate it from all other points. This way, the algorithm determines anomalies from normal data. Some points to consider regarding the Isolation Forest algorithm are that since Isolation Forests consist of splitting the dataset along a feature between the min and max of that feature it is inevitable that the outer points of a cluster will cause the prediction region to be misshapen. The algorithm will split the dataset until a point is isolated from all other points and set that whole region to either inlier or outlier depending on the average path length to this point. If a point on the edge of a cluster is considered an inlier, it will generate a region outward as inlier region. By adding contamination to the learning we can avoid this issue. Contamination is defined as having some portion of the training data be considered as outliers. These will naturally be the points on the edge of a cluster or with larger average path lengths. Introducing contamination will in essence label all points at the edge of the cluster as outliers, and will create a large region outward as an outlier region.

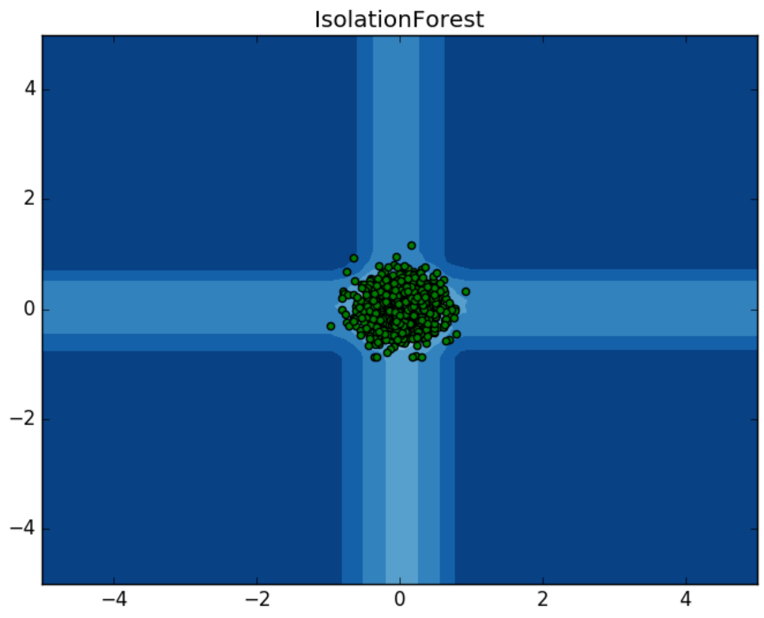


FIGURE 3.2: Example of Isolation Forest trained on two dimensional data in a cluster. The background color-scheme describes the decision function regions where lighter shades of blue are more likely to be inliers and darker shades are more likely to be outliers. This phenomenon is a product of the way the Isolation Forest algorithm works.

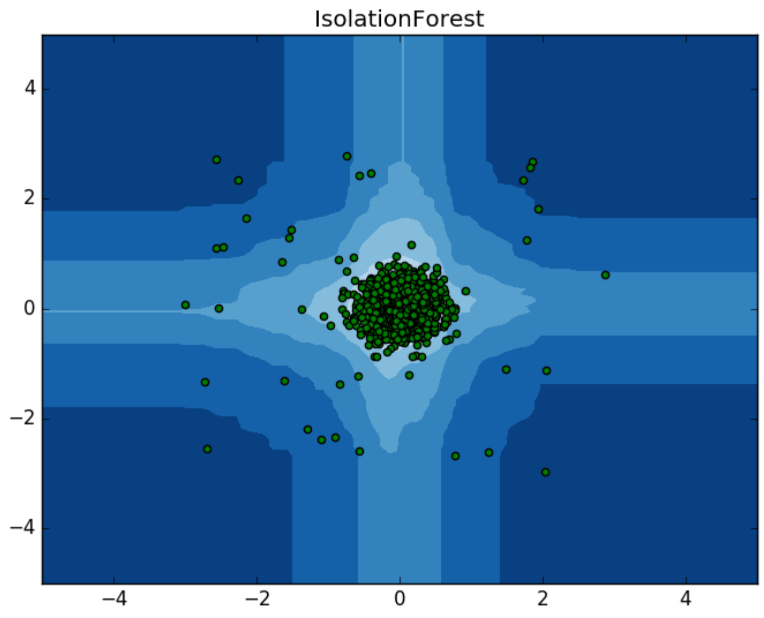
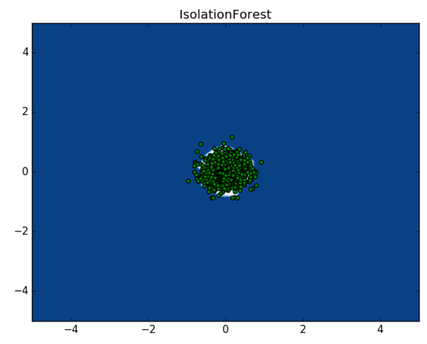
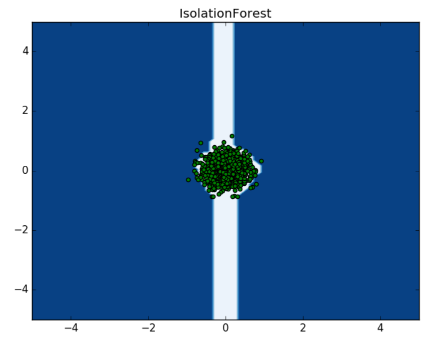


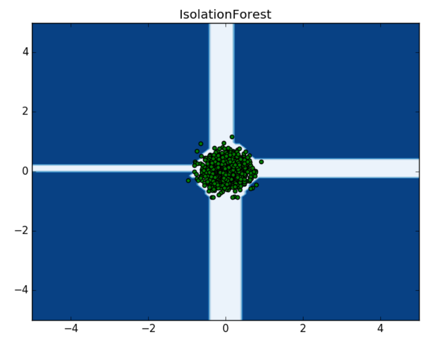
FIGURE 3.3: Example of Isolation Forest trained on two dimensional data in a cluster with some noise. The background color-scheme describes the decision function regions where lighter shades of blue are more likely to be inliers and darker shades are more likely to be outliers.



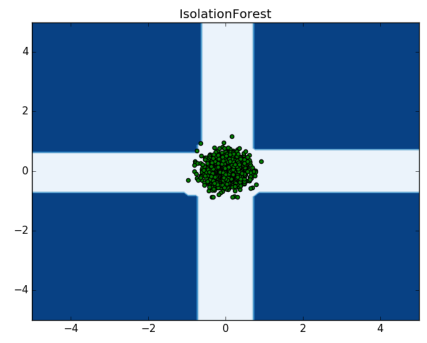
(A) contamination = 0.05



(B) contamination = 0.04

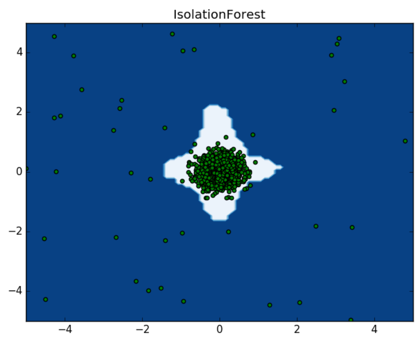


(C) contamination = 0.02

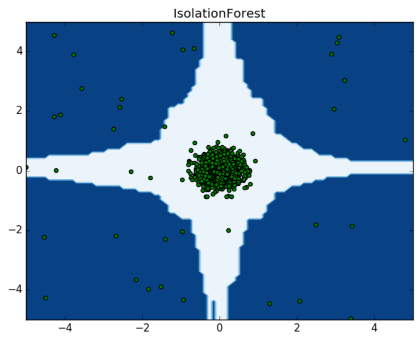


(D) contamination = 0.0

Figure 3.4: Isolation Forest in two dimensions fitted on the same training data but using four different values for the contamination measure. This dataset does not contain any outliers and the contamination parameter is set to zero. The plot shows the prediction region where white represents a prediction of inlier and blue is a prediction of outlier. The contamination is a measure of how many outliers are expected to be within the training set.



(A) contamination = 0.04



(B) contamination = 0.03

FIGURE 3.5: Isolation Forest in two dimensions fitted on the same training data with noise but using two different values for the contamination measure. The noise introduced leads to a contamination value of 0.4, and the two plots show the outlier regions when the contamination parameter is set to 0.4 and 0.3 for the same dataset.

The solution to the previously described issue regarding Isolation Forests would be to make sure the contamination value is set high enough to counter-act this phenomenon. Setting the contamination value higher will allow for a smoother decision region but it is also likely to exclude some true inliers.

3.3.2 RNN

The RNN built for this project is aimed at classifying single data points, as opposed to sequences. The network consists of the following layers:

- Input layer the size of the input features, in this case 38.
- Outer hidden layer with size 30, activation function *tanh*
- Inner hidden layer with size 7, activation function is a hard sigmoid
- Outer hidden layer with size 30, activation function *tanh*

- Output layer with size 38 and activation function sigmoid

This network is fully connected and was trained with a mean squared error loss function.

$$\text{mse} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where $\hat{Y}$ is the predicted value and Y is the target value from the network. The network used the stochastic optimiser "adam" with a learning rate of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and fuzzy factor $\epsilon = 10^{-8}$ [22]. The training data was completely filled with normal data and as each vector passes through the network, the weights are updated in a fashion to be more familiar with normal data. The structure of the network forces the weights to align in a way that from a compressed three-node layer, be able to reproduce a full 38 dimensional feature vector. After training, with the introduction of anomalous data, the network was expected to be unable to reproduce a full 38 dimensional vector from only three since the assumption is that anomalous data is very dissimilar to normal data. Hence, the mean squared error of a normal data point passing through the network is expected to be lower than an anomalous data point. This way the network is able to classify intrusion attempts based on some threshold of mean square error.
What can happen when a too slim selection of normal data has been used for training is that only that kind of data is classified as normal. For this method to work effectively, a very large variety of normal data must be used for training.

3.3.3 LSTM Autoencoder

To handle anomaly, and effectively intrusion detection of time series in a log file, the two previous methods will not do. The RNN is unable to learn patterns that span time, while an LSTM algorithm takes previous events into consideration when making classifications. This will prove vital when looking at a log file which may not raise any alarms for individual entries. The idea is to see whether the LSTM can identify malicious or unauthorised behaviour from unstructured data. The architecture of this network looks as follows:

- Encoding layer 1: input layer with a size dependent on the number of features extracted with tf-idf. In this case 137.
- Encoding layer 2: LSTM layer with 64 hidden units, activation function tanh and recurrent activation function hard sigmoid.
- Encoding layer 3: identical hidden LSTM layer stacked on top of the previous one but with only 3 hidden units for compression.

- Decoding layer 1: LSTM layer with 3 hidden units, activation function tanh and recurrent activation function hard sigmoid.
- Decoding layer 2: LSTM layer with 64 hidden units.
- Decoding layer 3: Time distributed dense layer with units equal to the number of features (137).

This LSTM autoencoder model was compiled using the mean squared error loss function and the stochastic optimiser "adam" with a learning rate of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and fuzzy factor $\epsilon = 10^{-8}$ [22]. This was written in Python, using Keras and Tensorflow to build the network.

The data processing for this problem includes the following steps:

- Collect packet capture files (PCAP) from network security test at FOI
- Convert PCAP to CSV file in order to read in as training data.
- Transform categorical data fields such as protocol and IP-address to numeric data by indexing with integers. This was written in a Python script.
- Splitting data into training and testing was done by choosing two sets where one was during a time of no attacks, and the other contained attacks. These two sets were further split into three sets named "training", "normal test" and "anomaly test". The "training" set contains only normal data, the "normal test" contains unseen normal data and "anomaly test" contains unseen data mixed normal and anomalous.
- Run tf-idf transform on the unstructured data fields in such a way that only terms with frequencies between 0.1 and 50 percent are kept.

The training and testing data came from two separate time frames during the penetration test at FOI. One was taken from a period of normal behaviour and the other from a period of an attack. To determine when this was, the activity logs of the red team from the FOI penetration test in 2011 were inspected. The indication of anomalous data is found when comparing the time stamps of scans and attacks with the time stamps of the captured packets. This will fail to give high accuracy results, but will give an indication of the algorithm's performance. The dataset extracted during the attack and used as anomalous data may contain normal behaviour as well.

For the feature extraction using tf-idf, some terms are excluded in the transform due to memory constraints. Terms which occur in fewer than 0.01 percent of documents and terms occurring in more than 50 percent of documents are ignored. The terms that occur in over 50 percent of documents are less likely to provide useful information due to being shared by most documents. While the terms that only occur in less than 0.01 percent of the documents may provide useful information, the feature vectors become too large to be used for training. This shall be covered in the results section.

Running the algorithm without memory overflow by generating batches from file:

- Creating multivariate sequential data takes up lots of memory and needs to be done in a generative manner. One sequential data point will be a three dimensional tensor and have the shape (1, sequence length, number of features). If there are many features or long sequences, a data set of millions of samples will quickly overload the memory. The sequential data was created with a sliding window approach with overlap.

  A smaller overlap allows for faster training whereas a larger overlap creates more representative data. The data was read in chunks at a time and a batch of sequential data was created. Feeding this into the LSTM model as batches of size 32 for fitting. The Pandas library for Python was used for streaming data and allowing to train on the whole dataset without reading it all to memory.

- The loss function used for training was the mean square error calculated for each sample sequence.

  $$\text{mse} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(Y_{ij} - \hat{Y}_{ij})^2$$

  where $\hat{Y}$ is the autoencoded sequence sample by the LSTM and Y is the true sequence. The index i runs over the features and index j runs over along the sequence. This is referred to as the reconstruction error.

Chapter 4

Results and Evaluation

4.1 Evaluation Metrics

The two metrics used to evaluate the models in this chapter are area under the ROC curve (AUC) and the F1-score. The ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR) while the AUC score is the area under this curve and provides a single value metric for the quality of the model. The AUC score is a common metric to evaluate the quality of binary classifiers, but is prone to overestimating a classifier on an imbalanced dataset. Due to the dependence of the true negatives in the AUC, a dataset containing many more negatives than positives can produce a high AUC score even though many false positives were raised. Since the F1-score takes both precision and recall into account it will give a fair evaluation even for an imbalanced dataset. Commonly, anomaly detection datasets consists of a large number of negatives and a small number of positives and by using both of these metrics, we can determine if the evaluation is affected by any imbalance in the dataset. Should both metrics follow a similar trend, it would be a strong indication that the evaluation is valid.

TPR = True Positive / (True Positive + False Negative)
FPR = False Positive / (False Positive + True Negative)

The F1-Score is the harmonic mean of Precision and Recall.

Precision = True Positive / (True Positive + False Positive)
Recall = TPR

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

4.2 DARPA KDD 99

The DARPA KDD 99 dataset is a common benchmark for intrusion detection and was used in this project to test the two algorithms of a Replicator Neural Network and Isolation Forest on network traffic without taking time dependencies into account. The dataset is labeled and structured, and the evaluation is done by using the F1-score and the AUC score. Anomaly detection datasets often have a large number of negatives (normal) and a small number of positives (anomalies). The DARPA dataset has an even number of positives and negatives but to be able to compare these results to other datasets, the F1-score is chosen as measure of performance.

4.2.1 RNN Parameters

The parameters of the RNN will affect the performance and here the sizes of the inner and outer layers are explored using box plots for statistical analysis. The line in the middle of the box plot represents the median, the edges of the box plot represent the first and third quartile and the whiskers outside represent the minimum and maximum if no outliers are present. The data points outside of the whiskers are considered outliers in the box plot.

Figure 4.1 shows that the F1-score for the RNN when the outer layer size varies from 4 to 39 with a step size of 5. The trend is that with a larger number of neurons, the F1-score increases. There is a jump between 29 neurons and 34 where the interquartile range drops significantly. When the number of neurons in the outer layer exceeds the number of input neurons (38) there is not a great increase in performance as this will no longer compress the data. Figure 4.2 shows the same thing as in Figure 4.1 but for the AUC score. It can be seen here that these two metrics do not correlate well, in fact it appears that the AUC score drops as the number of outer hidden neurons increases. The reason to include both metrics is to get a wider overview of the actual performance of the algorithm. When the F1 score is high there exists a threshold that separates the anomalies from the normal, but if the AUC is low at the same time it indicates that not all thresholds are good separators. Conversely, if the AUC score is high but the F1 score is low, there is a chance the dataset suffers from class imbalance. The F1 score does not take true negatives into account and is hence able to give a more accurate representation of performance on data containing a class imbalance of a large number of negatives and small number of positives.

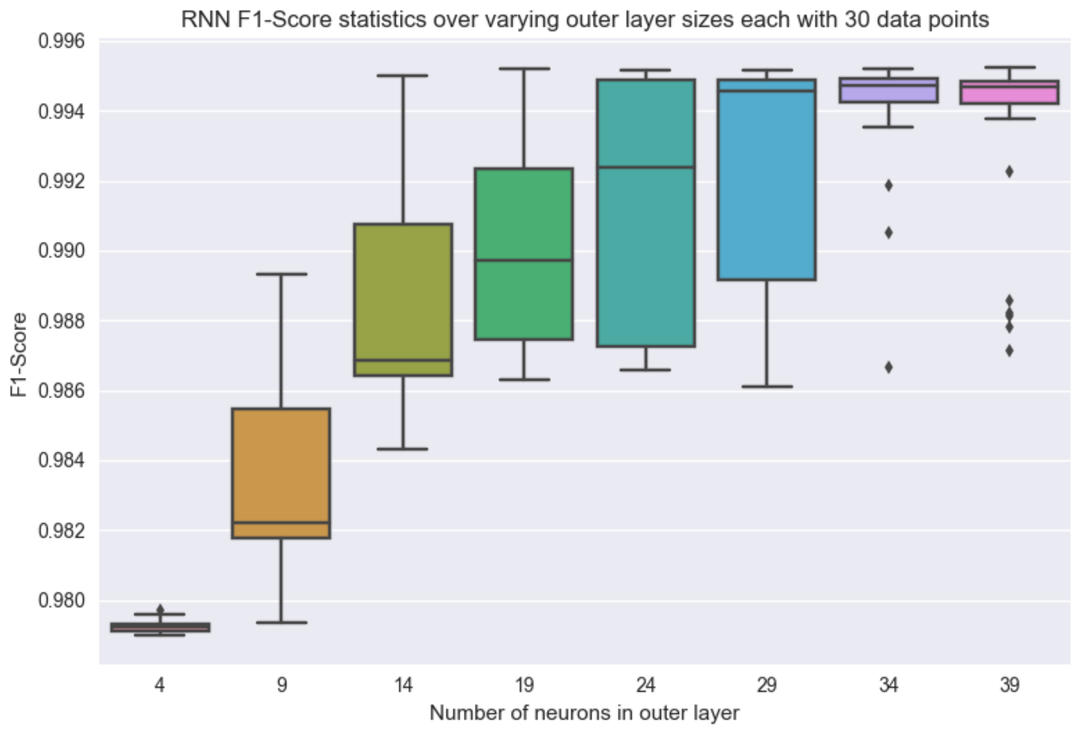


Figure 4.1: Statistical box plot of the F1-score with varying outer layer sizes. Each box plot is made up of 30 data points. The inner layer size was kept at 3 neurons.

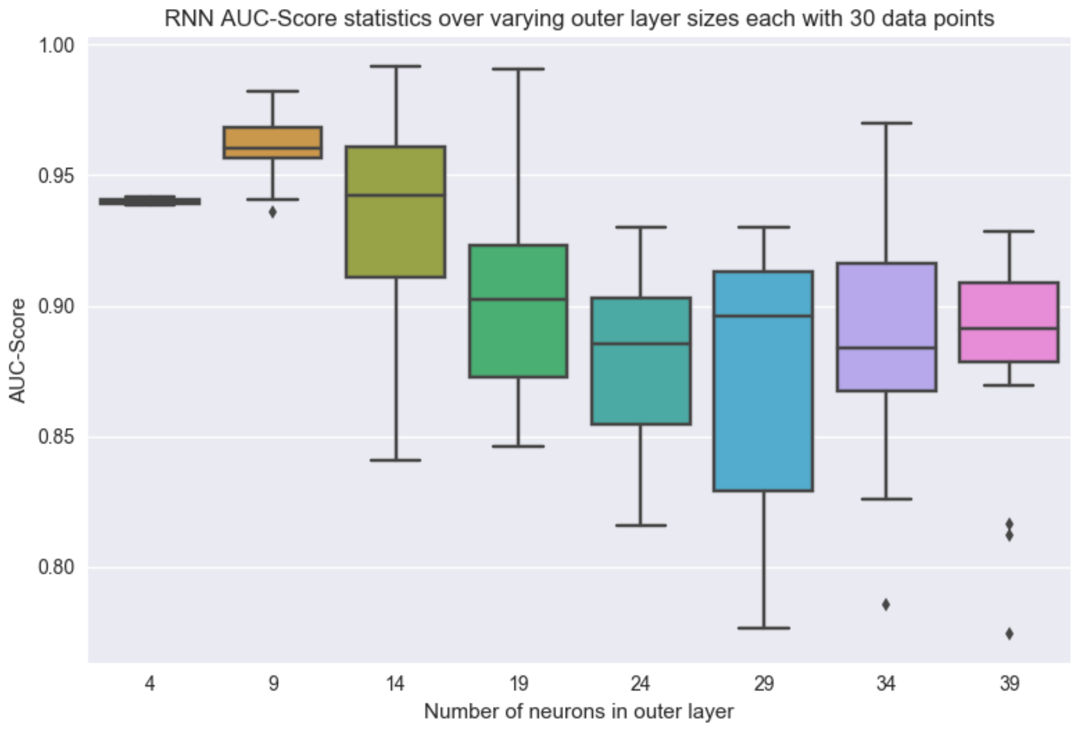


Figure 4.2: Statistical box plot of the AUC-score with varying outer layer sizes. Each box plot is made up of 30 data points. The inner layer size was kept at 3 neurons.

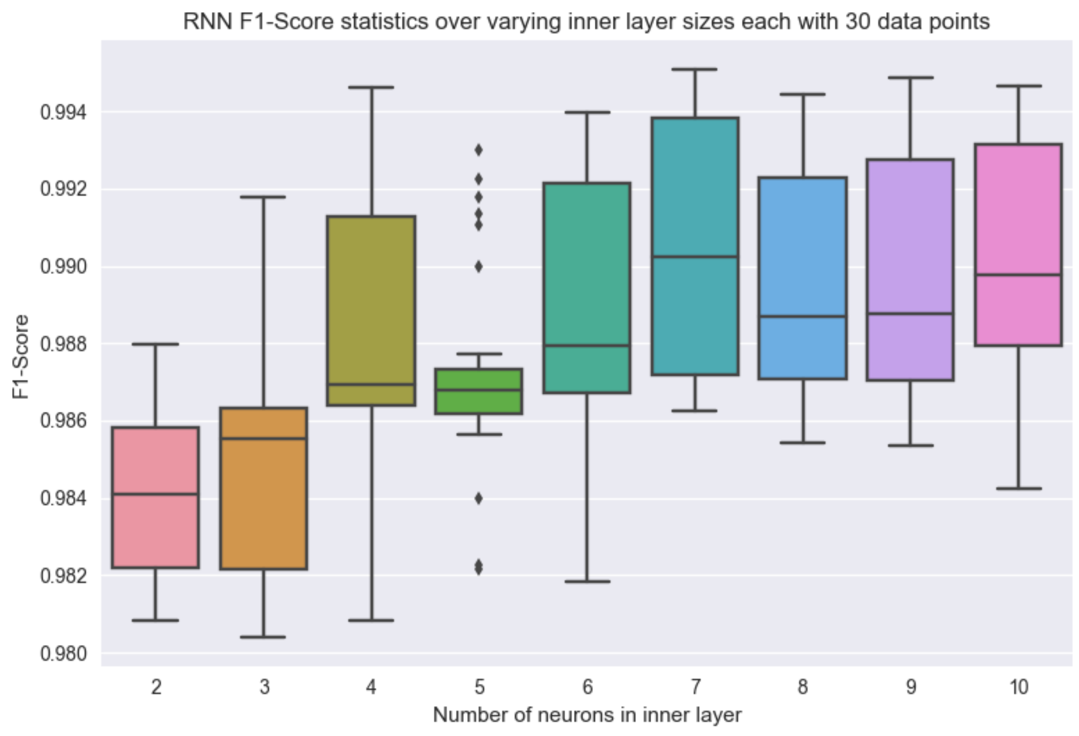


FIGURE 4.3: Statistical box plot of the F1-score with varying the inner layer size. Each box plot is made up of 30 data points. The outer layer sizes were kept at 30 neurons.

Figure 4.3 suggests that there is no great difference between the sizes of the inner hidden layer in the range 4 to 10 but that with only 2 or 3 neurons, the F1-score is slightly lower. This can be a sign that with only 2 or 3 neurons, the complexity of the model is not able to be encoded. The same is found in Figure 4.4 as the AUC score also does not have a persuading trend as the inner layer varies in size. Though the inter quartile range of layer sizes two and three are much smaller than the rest and this could be an indication of small inner layers performing better over many thresholds.

4.2.2 Isolation Forest Parameters

The parameters to be explored in the Isolation Forest algorithm are the number of trees in the forest, also called estimators, and the sampling size from the dataset to train each estimator.

Figure 4.5 shows that even a very small number of trees are able to detect a large part of the anomalies. The scale of Figure 4.5 is very small and the variation between the different box plots is also small. The final decision between anomaly and normal data is taken as an average over all the trees and seeing that only a few trees can classify the anomalies as unexpected. This suggests that the variance of misclassified anomalies is

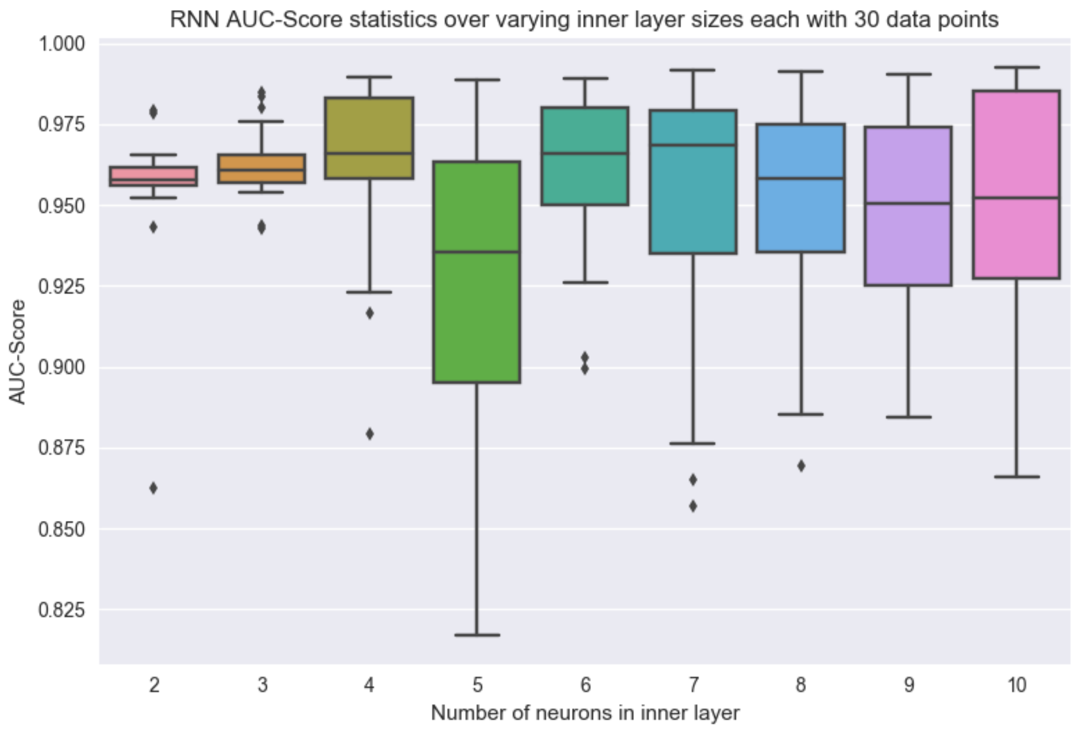


FIGURE 4.4: Statistical box plot of the AUC-score with varying the inner layer size. Each box plot is made up of 30 data points. The outer layer sizes were kept at 30 neurons.

very low, that they are inherently different from the normal data. Figure 4.6 corroborates Figure 4.5 and shows that all the box plots are in the high 99 percent region.

Figure 4.7 shows that there seems to be little difference between classifiers that sample from the training data and a classifier that uses the whole training data. The idea behind sampling from the training data is to avoid over-fitting as each tree is only trained on a subset of the training data. The fact that the performance is fairly constant over all sizes of sampling suggests that the model is unlikely to over-fit on the full data. Figure 4.8 shows that the AUC score also remains constant across all sample sizes.

The results seen in this section are all specific to the DARPA dataset and may vary for another dataset.

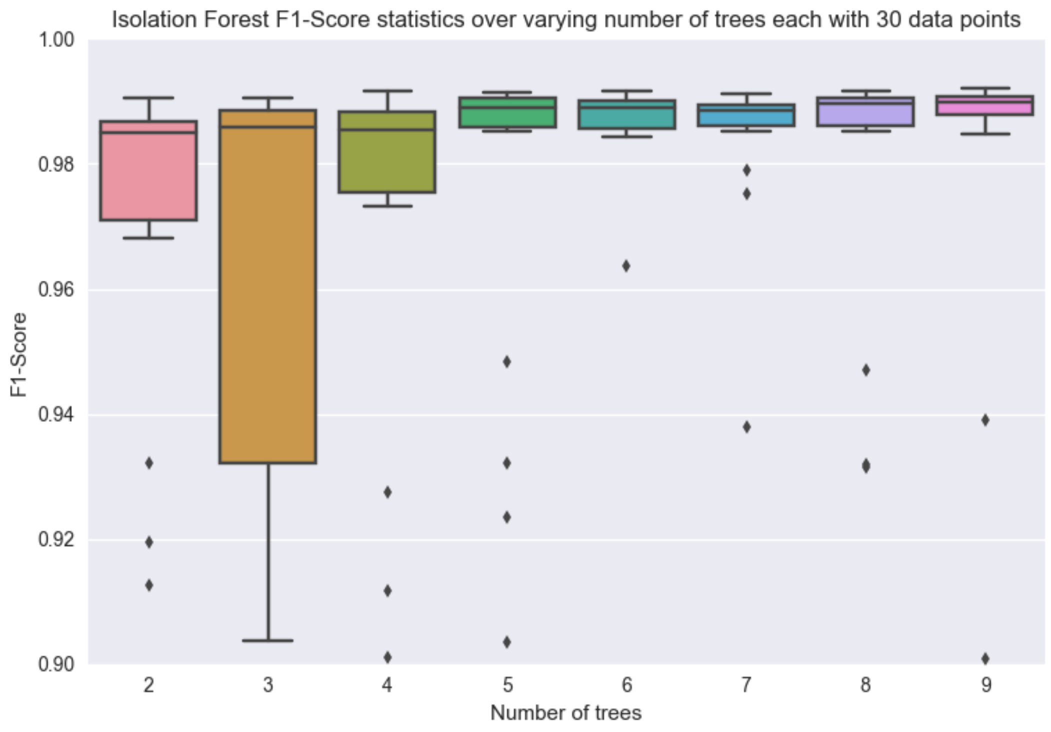


FIGURE 4.5: Statistical box plot of the F1-score with varying number of trees. Each box plot is made up of 30 data points.

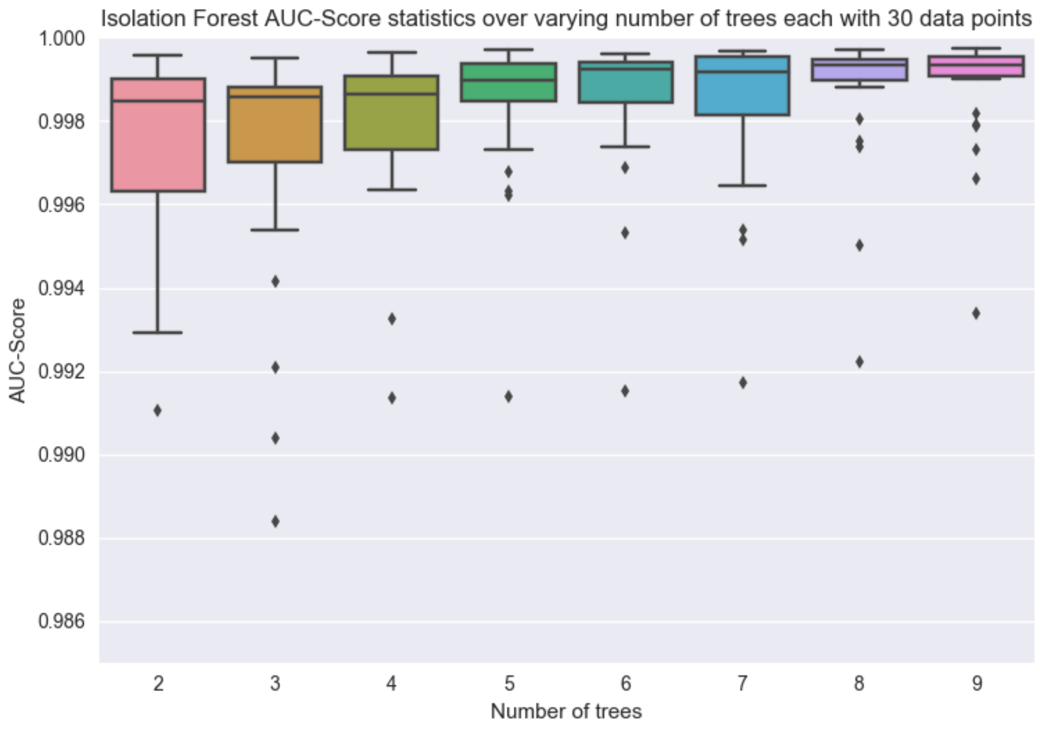


FIGURE 4.6: Statistical box plot of the AUC-score with varying number of trees. Each box plot is made up of 30 data points.

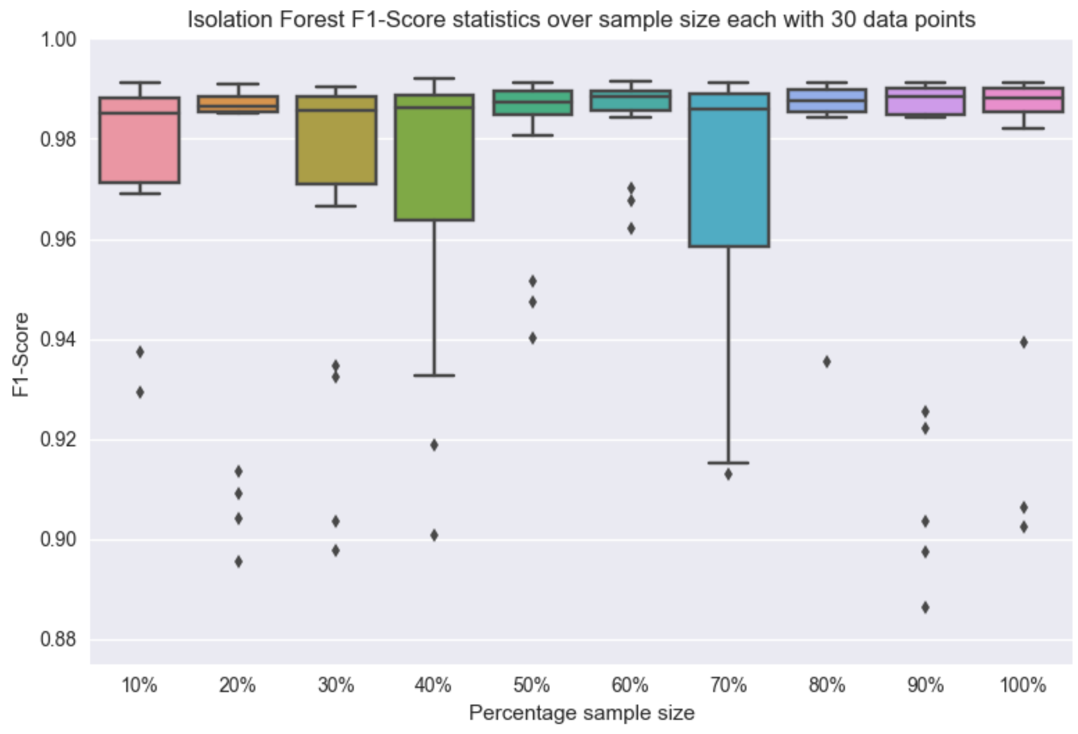


FIGURE 4.7: Varying the number of samples as a percentage to pick from the available training data and train each tree.

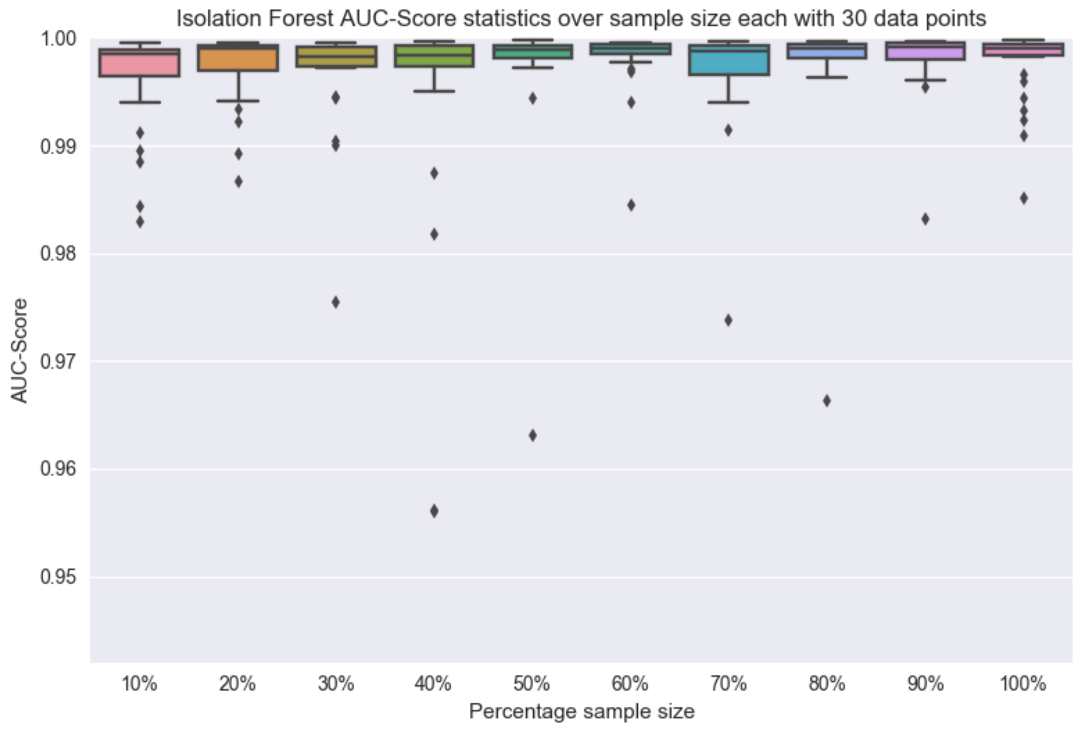


FIGURE 4.8: Varying the number of samples as a percentage to pick from the available training data and train each tree.

4.3 FOI 2011

The FOI 2011 dataset is unstructured time series event log data which makes it more interesting to examine than the labeled and structured DARPA set. The fact that this data is raw and comes from a real penetration test means that the results gained from evaluating this data can be implemented in organisations for practical use.
To begin evaluating the model, the reconstruction error for sequences coming from both normal and anomalous datasets are shown, where a sequence is defined as a list of contiguous network packets.

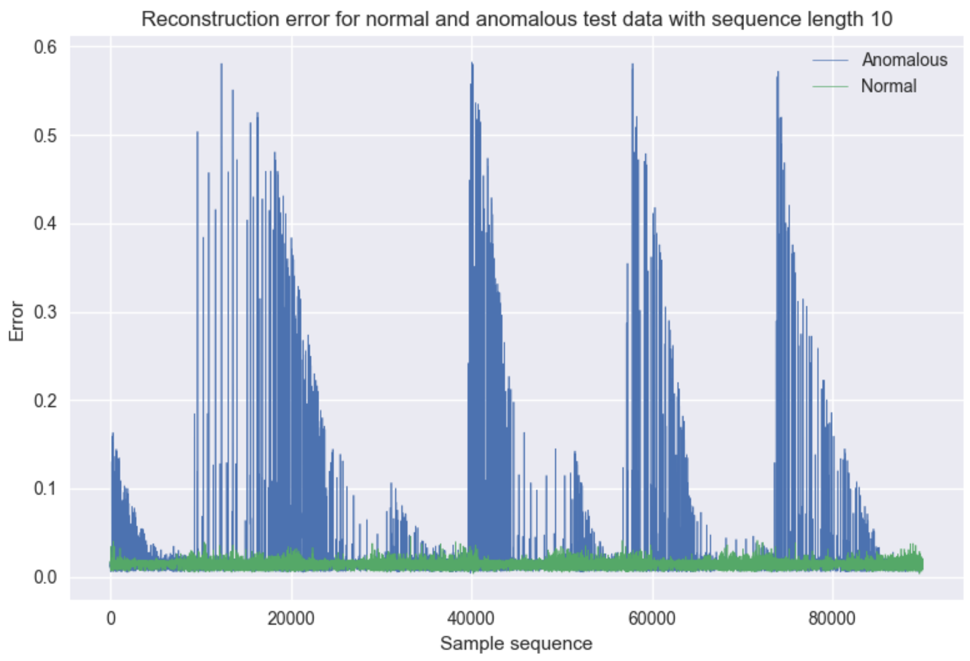


Figure 4.9: Reconstruction error for sequence length of 10

Figures 4.9, 4.10, 4.11, and 4.12 show the reconstruction error for the normal and anomalous test sets. The error for the normal set is seen to be significantly lower than the anomalous test error at all times, but for the lower sequences such as 10 and 100, there is much more overlap between the two test sets. This will make anomaly detection difficult as there is no threshold that perfectly separates them. What is shown in these figures is that the reconstruction error peak, this may be due to any of the following: the scan occurs periodically, normal behaviour is interspersed with anomalous data, or that the model does not detect the full scan but only sections. In any of these cases, the LSTM does detect anomalous activity as the peaks clearly show.

When evaluating the LSTM model on the normal test data it is expected to have a constant and low error, seeing as though it has been trained on similar data and should

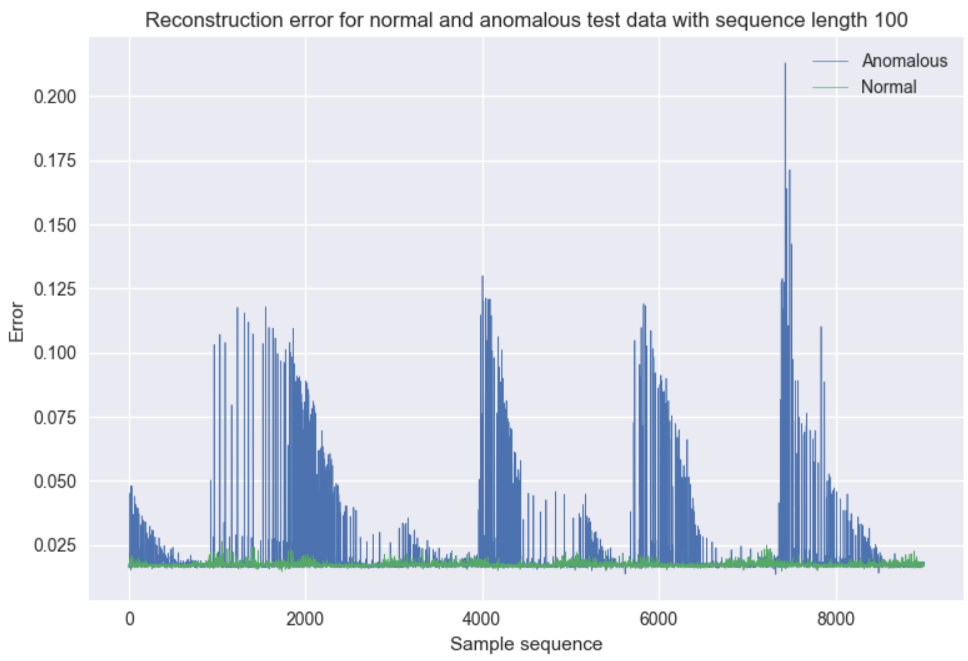


FIGURE 4.10: Reconstruction error for sequence length of 100

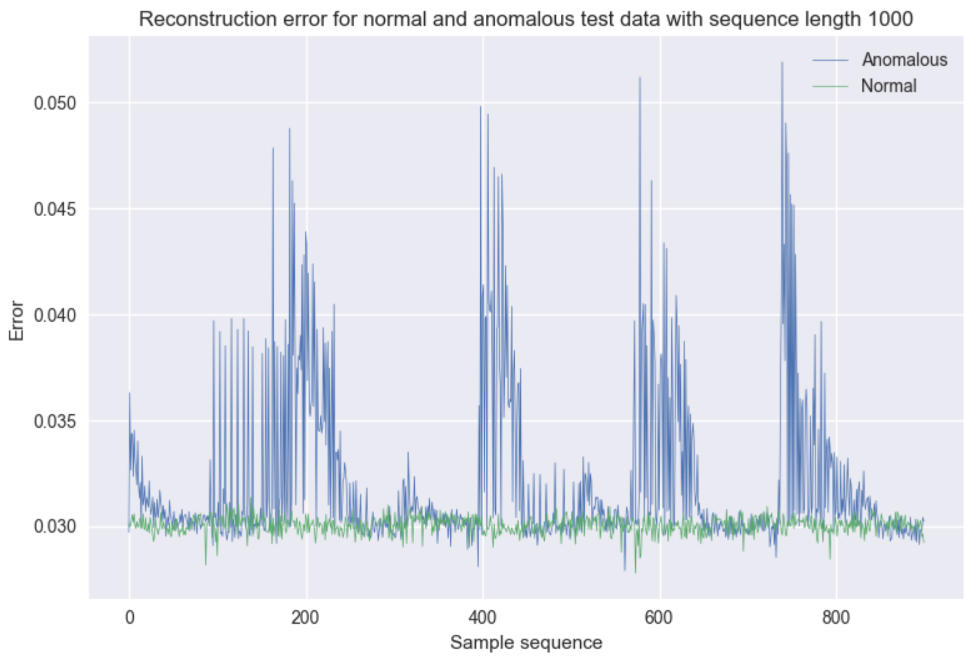


FIGURE 4.11: Reconstruction error for sequence length of 1000

be able to reconstruct the sequence once compressed. On the anomalous data set it is expected to have regions of high reconstruction error where scans or attacks take place, but also low regions in between where unseen normal data is found. Three subsets were extracted from the full dataset in such a way that two contain no anomalies and one spans a Simple Network Management Protocol (SNMP) scan. One of the subsets

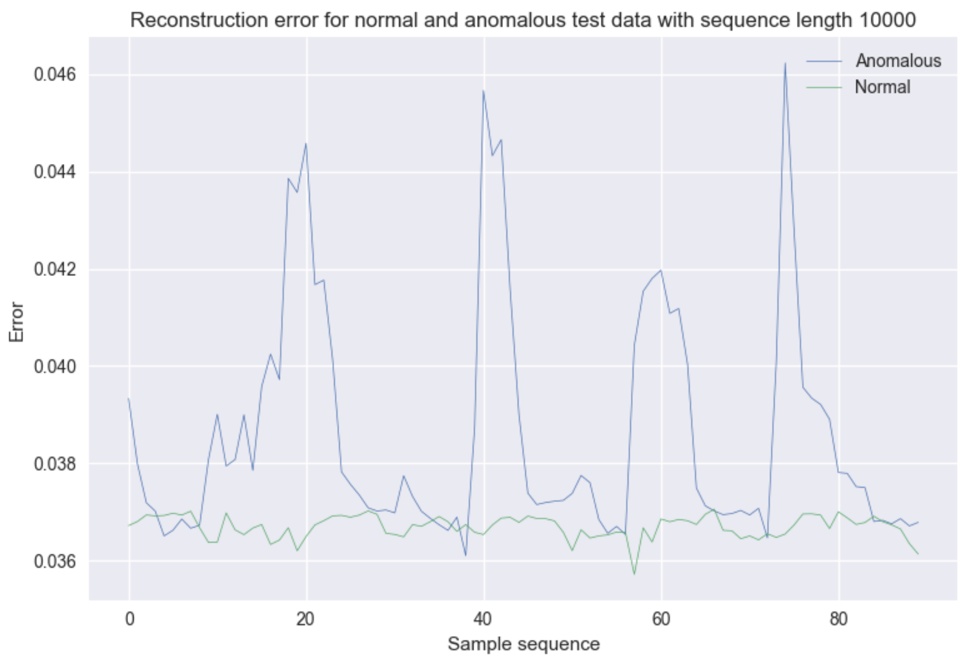


FIGURE 4.12: Reconstruction error for sequence length of 10000

containing no anomalies is used as training data, the other two subsets are used for testing. An anomaly is defined by a moving threshold over the reconstruction error. By varying the threshold both F1 and AUC scores are calculated. High scores in both of these metrics are attained when the threshold is able to separate anomalies from normal data. A high AUC score indicates there are multiple thresholds that can separate the data. Figure 4.13 shows the F1 and AUC scores when different sequence lengths are considered. The expected result is to see that many anomalies are not detected under short sequences. This is in agreement with Figure 4.13 as both F1 and AUC scores rise with longer sequences. The AUC score appears more sensitive to the length of the sequence whereas the F1 score is only affected once the sequence length becomes long enough, at 10^4.

Figures 4.14 and 4.15 show the F1 and AUC scores respectively in the form of box plots, each consisting of 20 data points. The features are extracted using tf-idf from the unstructured data components in the network packets. The parameters to vary are the minimum and maximum document frequency to include, where a document is defined as a single network entry. If a maximum document frequency of 0.5 is used, packets occurring in more than half of the dataset are ignored. One reason to do this is that if a packet is highly common, it may be less likely to provide insightful information on anomalies. On the other hand, having a minimum document frequency set at 0.01 will ignore packets that are seen less than one percent of the time. The uncommon packets are more likely to be connected to anomalies and hence, setting this parameter too high

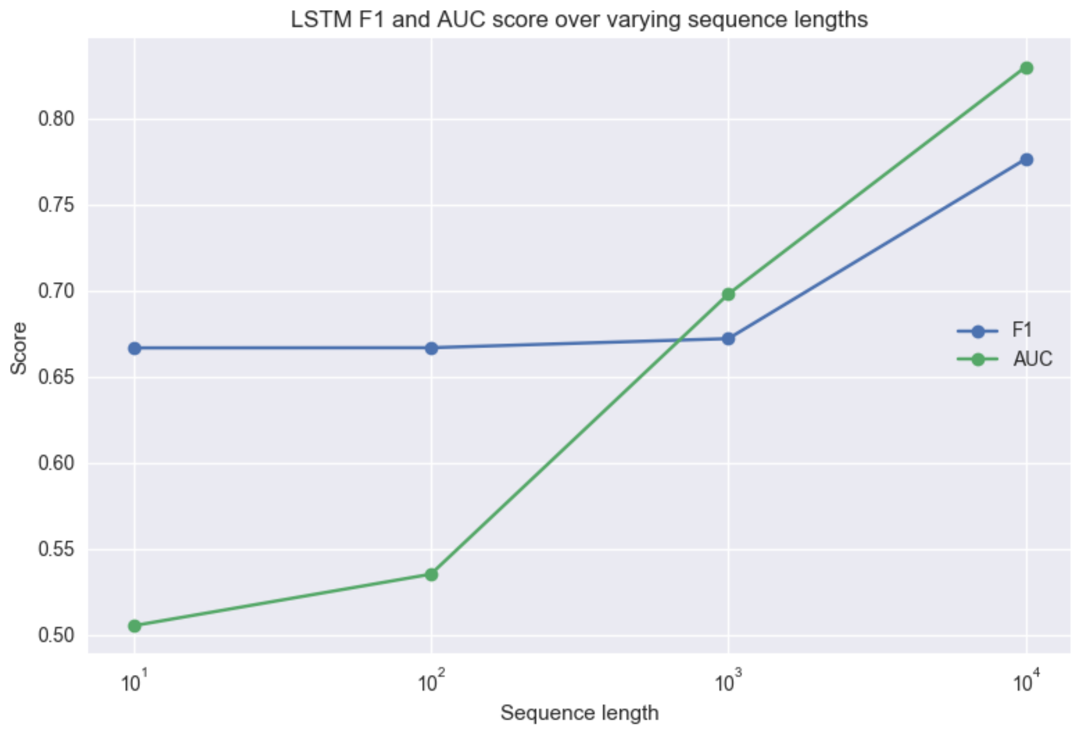


FIGURE 4.13: F1 and AUC scores plotted together to compare their behaviour to varying sequence lengths.

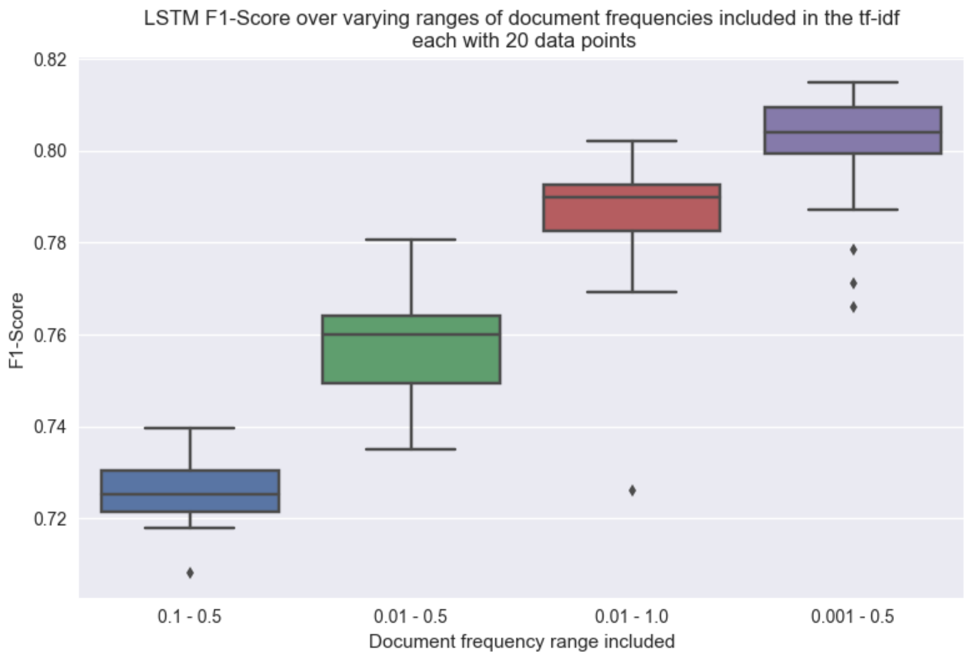


FIGURE 4.14: Box plot for the F1-score over document frequency ranges that are included during tf-idf feature engineering. The more documents that are included, the higher the feature map becomes. The shown ranges in the figure correlate to feature sizes 21,52,60,137.

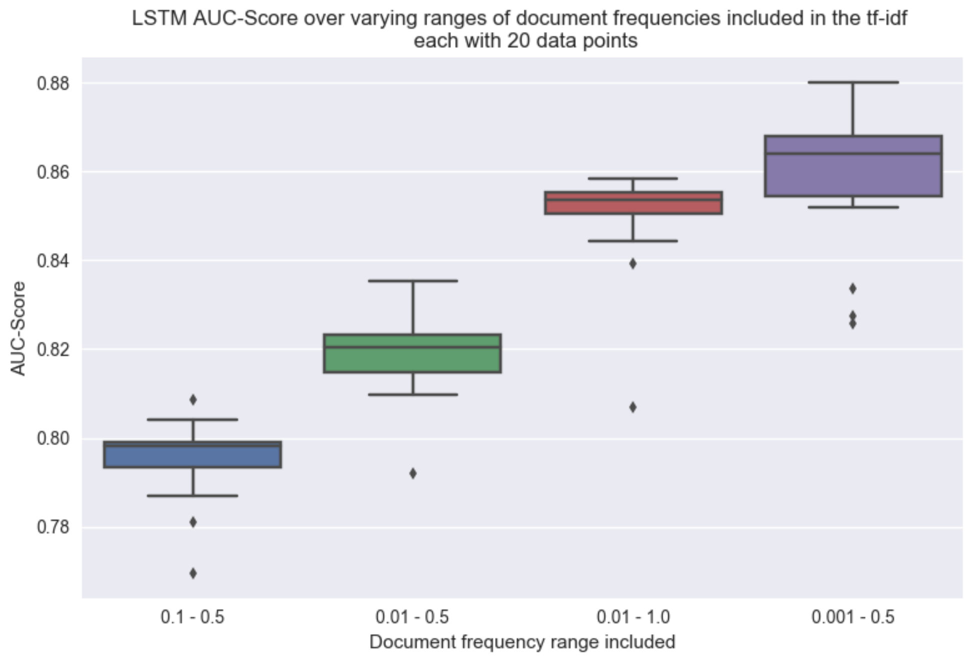


FIGURE 4.15: Box plot for the AUC-score over document frequency ranges that are included during tf-idf feature engineering. The more documents that are included, the higher the feature map becomes. The shown ranges in the figure correlate to feature sizes 21,52,60,137.

may lead to missing valuable information. That being said, including all packets will create a very large feature space and performance becomes an issue.

The results attained for the LSTM on the FOI dataset are on average much lower than the results from the RNN and Isolation Forest on the DARPA set. This is to be expected as the FOI dataset is unstructured. With 137 features extracted from the network packets, the algorithm reaches an F1 score of 0.8 and a ROC AUC score of 0.86.

Chapter 5

Conclusion

The question is whether an anomaly detection algorithm trained on unstructured time series data is able to effectively detect intrusions in network traffic. To answer the question, several steps need to be taken. Introducing a large number of algorithms which have the potential to solve the issue, and then choosing only a few to test is the first step. The chosen algorithms were Isolation Forests, Replicator Neural Networks and a Long Short-Term Memory Autoencoder. The Isolation Forest and RNN models were tested on a simple labeled and structured dataset (DARPA) to ensure their accuracy. The RNN can be viewed as a feed forward autoencoder neural network where the feature space is encoded into a much smaller space and then decoded to replicate the input. The idea with this with respect to anomaly detection is that if the model is trained solely on "normal" data, it will reconstruct such data with a lower error than "anomalous" data. Based on the results from both algorithms, a conclusion that may be drawn is that the data that represents these intrusions is not complex. A small neural network of [34,3,34] hidden nodes and an Isolation Forest with 5 trees are both able to reach an F1-score over 0.98. A different dataset may have a much more complex decision boundary or shape and it is not certain that these models with these parameters would detect those intrusions. Taking the idea of the Replicator Neural Netowrk and expanding the feed forward neural network to include recurrent connections we arrive at the LSTM autoencoder which is recurrent but also does well with handling the vanishing gradient problem. Such a network is better suited for dealing with long time series and hence finding anomalies within the network traffic. In order to train this model, the unstructured data needs to be represented numerically first. This was done with the use of the "term frequency - inverse document frequency" method which gives each term in a document a relevancy score depending on its frequency and the inverse of the document frequency. The results show that these two methods together form an intrusion detection system. The resulting plot of the errors calculated over normal and anomalous data were partly as expected, but also contained unexpected peaks. The scan to be detected was found by a moving threshold over the reconstruction error of the test set. F1 and ROC AUC scores were calculated to evaluate the performance based on varying parameters. Using the knowledge gained from exploring parameters in the RNN, the autoencoder structure was easily adjusted to the LSTM network. In the end, the LSTM autoencoder reached an F1 score of 0.8 and an ROC AUC score of 0.86 using 137 features from the unstructured data. While these results are lower than for the benchmark DARPA dataset, they are still high enough to conclude that an Autoencoder structure coupled with LSTM neurons are able to detect intrusions in sequential unstructured data. The results provided here are specific to the FOI dataset and may not generalise elsewhere. The strong point of the proposed solution is the idea of a LSTM autoencoder coupled with numeric representation of unstructured data. The LSTM network may catch incidents that a feed forward network would not due to a vanishing or exploding gradient in long sequences. Using the tf-idf method and cleaning up the result by removing both common and very uncommon terms allows for a compact feature space which is still sufficient for training. These are terms occurring in more than 50 percent of documents and terms occurring in less than 0.1 percent of documents and resulted in 137 features. The weak point does not lie in the solution itself, but in the data and training. Due to memory and time constraints, only a small subset of the available training data was used. With more powerful technology and lower time constraints, more training data could be used, more terms from the tf-idf matrix can be included and longer sequences can be evaluated. These factors are likely to provide a more accurate result.

Bibliography

[1] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. Intrusion detection: A survey. In *Managing Cyber Threats*, pages 19–78. Springer, 2005.

[2] Little bang.
`https://littlebang.wordpress.com/2009/11/19/the-missing-black-box/`.
Accessed: 2017-05-08.

[3] Deep neural network's precision for image recognition, float or double.
`https://stackoverflow.com/questions/40537503/`
`deep-neural-networks-precision-for-image-recognition-float-or-double/`.
Accessed: 2018-03-12.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[6] Smia 2011 system infrastructure. URL
`ftp://download.iwlab.foi.se/dataset/smia2011/System_documentation/`.

[7] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 38–44. IEEE, 2015.

[8] Rebecca Bace and Peter Mell. Nist special publication on intrusion detection systems. Technical report, BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, 2001.

[9] Alfonso Valdes and Keith Skinner. Adaptive, model-based monitoring for cyber attack detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 80–93. Springer, 2000.

[10] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian event classification for intrusion detection. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 14–23. IEEE, 2003.

[11] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] Nong Ye, Yebin Zhang, and Connie M Borror. Robustness of the markov-chain model for cyber-attack detection. *IEEE Transactions on Reliability*, 53(1): 116–123, 2004.

[14] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[15] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145. IEEE, 1999.

[16] Mark Crosbie, Gene Spafford, et al. Applying genetic programming to intrusion detection. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 1–8. Cambridge, MA: MIT Press, 1995.

[17] Anup K Ghosh, Christoph Michael, and Michael Schatz. A real-time intrusion detection system based on learning program behavior. In *International Workshop on Recent Advances in Intrusion Detection*, pages 93–109. Springer, 2000.

[18] Simon Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

[19] Simon Hawkins, Hongxing He, Graham J. Williams, and Rohan A. Baxter. Outlier detection using replicator neural networks. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK 2000, pages 170–180, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44123-9. URL `http://dl.acm.org/citation.cfm?id=646111.679466`.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 413–422. IEEE, 2008.

[21] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[23] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128. IEEE, 1996.

[24] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[25] Eleazar Eskin, Wenke Lee, and Salvatore J Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 165–175. IEEE, 2001.

[26] Susan M Bridges, Rayford B Vaughn, et al. Fuzzy data mining and genetic algorithms applied to intrusion detection. In *Proceedings of 12th Annual Canadian Information Technology Security Symposium*, pages 109–122, 2000.

[27] Lotfi A Zadeh. Fuzzy sets. In *Fuzzy Sets, Fuzzy Logic, And Fuzzy Systems: Selected Papers by Lotfi A Zadeh*, pages 394–432. World Scientific, 1996.