ZERO-DAY ATTACK DETECTION WITH DEEP LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BERNA ÇAKIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2019

Approval of the thesis:

ZERO-DAY ATTACK DETECTION WITH DEEP LEARNING

submitted by **BERNA ÇAKIR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assist. Prof. Dr. Fuat Akal
Computer Engineering, Hacettepe University _____

Date: 29.08.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Berna Çakır

Signature:

ABSTRACT

ZERO-DAY ATTACK DETECTION WITH DEEP LEARNING

Çakır, Berna
Master of Science, Computer Engineering
Supervisor: Assist. Prof. Dr. Pelin Angın

August 2019, 78 pages

The rise of the IoT paradigm in the past decade has resulted in an unprecedented number of zero-day attacks launched against IoT systems, which are capable of causing major damages. Deep learning has recently become a popular technique for many learning tasks including intrusion detection, with high potential to detect zero-day attacks in addition to ones with well-known signatures. In this thesis, we analyzed the efficacy of supervised and unsupervised deep learning algorithms for detecting zero-day attacks. We experimented with different neural network architectures including fully connected, recurrent and temporal convolutional models. The proposed deep learning models were proven to be effective in intrusion detection with achievement of 95.3% classification accuracy and 97% f1-score. The models were tested on datasets created using the same environment with the training dataset as well as datasets created in different environments through transfer learning. The tests on the datasets, which were created in different environments showed that deep learning algorithms are capable of detecting some of the attacks with low false positive rates.

Keywords: Intrusion Detection, Deep Learning, Neural Networks

ÖZ

DERİN ÖĞRENMEYLE SIFIRINCI GÜN SALDIRI TESPİTİ

Çakır, Berna
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Danışmanı: Dr. Pelin Angın

Ağustos 2019, 78 sayfa

Nesnelerin İnterneti paradigmasının son yıllardaki yükselişi, bu sistemlere yönelik büyük hasarlara neden olabilecek benzeri görülmemiş sayıda sıfırıncı gün saldırına alan açmıştır. Derin öğrenme son zamanlarda, tanınmış imzalara sahip olanlara ek olarak sıfırıncı gün saldırılarını tespit etme potansiyeli yüksek, popüler bir teknik haline gelmiştir. Bu tezde, gözetimli ve gözetimsiz derin öğrenme algoritmalarının sıfırıncı gün saldırılarını tespit etmedeki etkinliğini analiz ettik. Analizleri farklı derin öğrenme modelleri üzerinde yaptık ve bu algoritmaların performanslarını birbirleri ile kıyasladık. Test edilen modeller derin öğrenme algoritmaları ileriye dönük sinir ağları, özyineli sinir ağları ve evrişimli sinir ağlarıdır. Test sonuçları, %95.3 doğruluk puanı ve %97 f1-puanı ile güdümsüz derin öğrenme metodlarının saldırı tesbitinde başarılı olduğunu gösterdi. Eğitim seti ile aynı ortamda üretilen test setlerinin yanı sıra eğitim setinden farklı ortamlarda oluşturulmuş test setleri de derin öğrenme yöntemlerini test etmek için kullanıldı. Yapılan bu testler derin öğrenme yöntemlerinin farklı ortamlarda oluşturulmuş setlerdeki her saldırıyı tespit edemese de bazı saldırıları düşük yalancı pozitif oranları ile tespit edebildiğini gösterdi.

Anahtar Kelimeler: Bilgi Güvenliği, Derin Öğrenme, Yapay Sinir Ağları

To family and friends

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Pelin Angın for giving me the opportunity to work with her during my thesis. Thanks to her guidance, friendly and supportive attitude, I was able to complete this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

SYMBOLS	
IDS	Intrusion Detection System
SIDS	Signature Based Intrusion Detection System
AIDS	Anomaly Based Intrusion Detection System
FCN	Fully Connected Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
TCN	Temporal Convolutional Network
BPTT	Backpropagation Through Time
tBPTT	Truncated Backpropagation Through Time

CHAPTER 1

INTRODUCTION

1.1. Intrusion Detection

Intrusion is defined as any kind of threat to information confidentiality, integrity or availability such as unauthorized use, alteration or destruction of the information systems. An Intrusion Detection System (IDS) is a system whose purpose is to identify malicious network activities to assist in maintaining computer security. [1]

In today's age, networks are used for many tasks for which security is essential. Applications such as online banking and shopping systems, online businesses, emails, cryptocurrencies, even military operations rely on computer networks. However, as the dependence on computer network increases, the threats to networks also increase. The increasing number of threats makes computer security a task of highest importance. According to [2], in 2018, cyber-attacks caused losses more than $45 billion worldwide.

To fight with cyber-attacks, several hardware and software solutions including antivirus systems, firewalls etc. are developed. One of these solutions is intrusion detection systems (IDS). IDSs are hardware or software systems, which monitor the network to detect malicious activities. Figure 1.1. shows diagram of an intrusion detection system which monitors the traffic between the network which is protected by the IDS system and rest of the Internet. IDS raises an alert if it detects malicious activities [3].

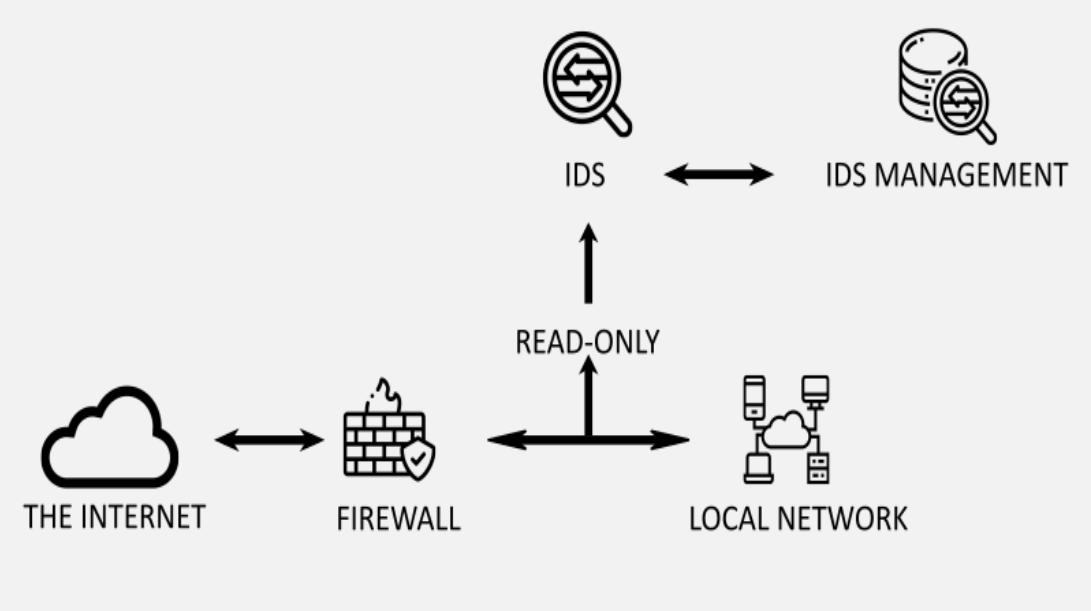

Figure 1.1. Intrusion Detection System

Based on the methods used for intrusion detection, IDSs can be divided into two categories: Signature-based Intrusion Detection System (SIDS) and Anomaly-based Intrusion Detection System (AIDS). SIDS keep a database of existing attack signatures and use pattern matching techniques to identify attacks with matching signatures. SIDS are very effective on detecting known attack vectors with low false positive rates. However, they cannot detect attacks, which are not already stored in their signature database [4].

Anomaly-based Intrusion Detection Systems use heuristic methods to create a model of the network defining legitimate patterns and user behaviors as baseline. The model is then used to identify the deviations from the baseline model to find potential intrusions. AIDS rely on the assumption that the behavior of malicious actors are different from the typical user behaviors. The main advantage of this method is that, with this model even unknown attacks can be detected. However, since anomaly-based methods classify any deviations from the normal as intrusion, unseen system behaviors can be detected as intrusions, which can lead to high false positive rates.

1.2. Zero-Day Attacks

One of the most challenging tasks of IDS is detection of zero-day attacks. A zero-day attack is an attack, which exploits a previously undiscovered network vulnerability.

The life of a zero-day attack is shown in Figure 1.2 from the day the software with the vulnerability is released by the software maker until the update to fix the vulnerability is fixed and distributed. The life of a zero-day attack starts with the discovery of a vulnerability in a piece of software. Then, the vulnerability is used to create an exploit and attack the targets. After the initial attacks are executed, the vendors which released the vulnerable software find out about it and create a patch to fix it. The vulnerability does not stop being a threat until the patch is released and distributed.

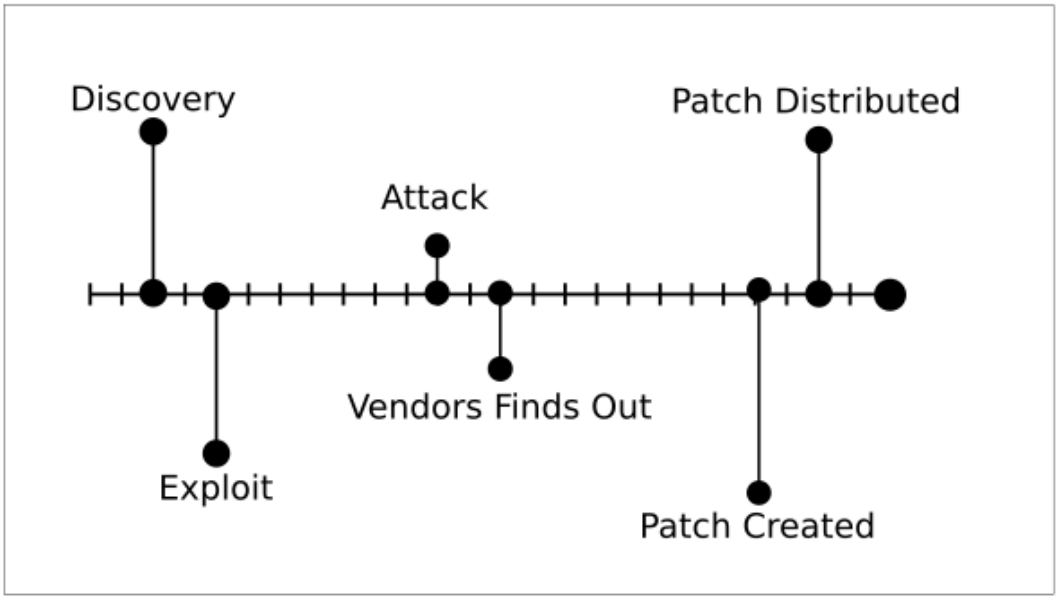

Figure 1.2. The Life of a Zero-Day Attack *[5]*

Since finding vulnerabilities in software requires time and skills, zero-day attacks are mainly used as targeted attacks against high-value information systems. However, after that, the vulnerability becomes known and variants of it affecting the general public are created. After a vulnerability is disclosed, the number of attacks using the

same exploit increases 100000 times. The exploit continues to prevail as a threat until all hosts are patched for the vulnerability, which takes between 20 days to 30 months with an average of 312 days. [6]

A zero-day attack on a selected target usually is not a single attack, but a sequence of attacks, which includes exploration to find weaknesses, gaining unauthorized access to the target, gaining elevated privileges on the target to use, modify or delete information. These sequences of attacks are known as multi-step attacks or Advanced Persistent Threats (APTs). [6]

Signature based intrusion detection methods cannot detect zero-day attacks since no signatures exist for such attacks prior to the attack. Because of this, with the increase of the zero-day attacks, SIDS have become less and less effective as an IDS method [4]. On the other hand, anomaly based intrusion detection systems are capable of detecting attacks which they have not seen before such as zero-day attacks.

1.3. Motivation

Anomaly based intrusion detection algorithms are preferred over signature-based intrusion detection algorithms when it comes to detection of unknown attacks.Since Denning proposed the hypothesis that security violations can be detected as abnormal patterns of system usage in 1987, several artificial intelligence techniques have been used to develop intrusion detection systems.[7] Out of these methods, deep learning is a relatively new field. The aim of this study is to compare deep learning based anomaly detection algorithms for intrusion detection.

Although several works exist on the topic, because of the problems explained below it is hard to compare existing anomaly detection methods. The first problem is the way the training and testing datasets are selected. There are several works which do not use the testing dataset, opting for methods such as cross validation or works, which uses only subsets of the test set. These works report high accuracies many of which are above 99%. [8]–[12]. These approaches do not show capabilities of the intrusion detection systems since very high accuracies on training dataset can be obtained by

overfitting the training dataset with the absence of the test set. The second problem with existing works is the way the results are reported. In existing intrusion detection datasets, Denial of Service (DoS) attacks, whose purpose is to flood the network with useless packages, are more frequent than other attack types. In addition to that, such attacks are easier to detect without complex intrusion detection methods. Thus, if individual attack detection rates for different attack types are not reported, it is not known whether a technique is capable of detecting different types of attacks or only a few frequent attacks. In addition to that, most of the existing works do not report the performance of their model on detecting previously unknown attacks.

Lastly, most papers do not report the zero-day attack detection capabilities of their models. To test the zero-day attack detection capabilities of the models, the detection rates for attacks that do not exist in the training set must be reported.

The contribution of this thesis is to apply existing deep learning techniques to the intrusion detection field and measure and compare their performances in areas especially in detection of zero-day attacks. The algorithms are further tested with transfer learning by training the model in a dataset and testing it by using another, completely unrelated dataset. This allows to compare the behaviors of the deep learning-based anomaly detection algorithms in an unseen environment.

1.4. Outline of The Thesis

The remainder of this thesis is organized as follows: In chapter 2, the previous works about deep learning-based intrusion detection are summarized. In chapter 3, the deep learning concepts relevant to this thesis are summarized. In chapter 4, how the deep learning methods are used for the intrusion detection field are explained. In chapter 5 the results of the experimental study are shown. Chapter 6 provides information about the transfer learning experiments. Chapter 7 summarizes the findings of this thesis and explains the future work directions.

CHAPTER 2

ANOMALY BASED INTRUSION DETECTION

2.1. Introduction

This section describes previous works on anomaly-based intrusion detection systems. Anomaly-based intrusion detection systems detect network anomalies via analyzing network patterns and detecting activities that do not match the normal patterns. AIDS must thoroughly understand the characteristics of the normal data in order to be able differentiate the abnormal patterns.

Machine learning is a subfield of artificial intelligence, which aims to develop algorithms and mathematical models that can learn from sample data without explicit instructions. The models developed using machine learning extract patterns and use inference to learn the characteristics of the sample data to make predictions about unseen data. Machine learning has been applied extensively to the anomaly-based intrusion detection field and has proven to be very successful.

2.2. Supervised and Unsupervised Training

Based on the method used for training, machine learning methods can be classified into two categories: supervised training and unsupervised training.

In the supervised method, the mathematical model is trained using a fully labelled dataset. Supervised methods can be further divided into two categories: Regression and Classification. Regression algorithms are used when the output is from a continuous range, while classification algorithms are used when the outputs are discrete categories.

Decision Trees, k-nearest neighbor (K-NN), naïve Bayes, SVM and Artificial Neural Networks are some of the most popular supervised machine learning algorithms. All

of these algorithms have been used in the intrusion detection field with varying degrees of success. [13]

Unsupervised methods are trained using unlabeled data. In general, unsupervised algorithms cluster the input into categories based on commonalities of the data points. Unsupervised algorithms are used when a labelled dataset is not available. Clustering algorithms such as K-means clustering and hierarchical clustering are examples of unsupervised machine learning algorithms. [1]

Sometimes, it is far easier to collect data belonging to one class than collecting data for other classes. For example, in the anomaly detection case collecting non-anomalous data is easier than collecting anomaly data since anomalies are rarer. This is also the case for the intrusion detection field, as it is easier to collect network data without attacks than collecting network intrusion data. In that case semi-supervised methods can be used to start with partially labelled data, learn characteristics of the normal data and find deviations from the normal. One-Class SVM and Autoencoders are examples of algorithms that can work in a semi-supervised fashion.

Although machine learning algorithms have proven to be successful in learning from the data without explicit instructions, the performance of the machine learning algorithms heavily depends on the representation of data. Each piece of information in the representation is called a feature and in order for machine learning algorithms to perform well, the features must be selected carefully using domain specific knowledge. Deep learning is a subfield of machine learning, which can solve this representation problem. Deep learning models have a multi-layered architecture, which allows them to do feature selection on their own. The first layers of the deep models learn simpler representations from the data, which are then combined to create complex representations. [14]

Currently, deep learning is a very powerful tool both for supervised and unsupervised learning tasks including anomaly-based intrusion detection.

2.3. Deep Learning Based Intrusion Detection

Deep Learning is one of the most used machine learning methods in the intrusion detection field and has proven to be successful. Deep neural network classification methods can be used for supervised training, while autoencoders can be used for unsupervised training. Deep learning methods can extract higher level features from the given data with high a success rate. In the intrusion detection field, similar to other anomaly detection fields, the samples in the dataset are skewed towards the normal data, which in turn makes the network also biased toward the normal data. As a result, deep neural networks have lower success in detection of less frequent attacks. Deep neural networks also have convex optimization functions, which means they can converge on a local minimum, and the global minimum cannot always be found.

Deep learning methods have been used in the intrusion detection field since 2008, though the datasets used were much smaller at that time. In 2008, [15] proposed a feed forward neural network to create an IDS, which uses the back-propagation algorithm to train the network. They ran tests on a dataset extracted from DARPA KDD'98 TCP dumps. The network had 76% accuracy rate on the test dataset consisting of 100 samples. Mukkamala, proposed a hybrid approach combining neural networks and SVM to detect intrusion [16]. Xue et al. proposed to modify recurrent neural networks for intrusion detection. The KDD'98 dataset was used for testing and 97% accuracy was reached on a 200-sample test dataset [17]. Besides, Skaruz also successfully applied recurrent neural networks to detect SQL-based attacks [18].

Recent improvements in available hardware and software allowed usage of larger datasets along with deeper deep learning models. Both supervised and unsupervised deep learning methods have been applied on intrusion detection datasets.

Deep neural network classifiers are used in many studies in the field of intrusion detection. [19] checked the potential capability of deep neural network as a classifier for different types of intrusion attacks on the KDD'99 dataset. A multi-layer feed forward network was trained using the KDD'99 dataset. The paper reported a

99% training accuracy and showed that DNN can achieve a lower training error than SVM. However, this paper did not report the testing accuracy using the testing dataset of the KDD'99 dataset. [20] built a deep neural network classifier model for an intrusion detection system in an SDN environment and trained the model with the NSL-KDD Dataset. They just used six of the 41 features of NSL-KDD, which can be easily obtained in SDN. The results showed that although there are algorithms, which have higher testing accuracy than DNN, DNN is still able to get an acceptable detection rate only using 6 features.

Recurrent neural networks were also used in the intrusion detection field. [21] presented an on-line deep learning method for intrusion detection using an LSTM network trained using the Real Time Recurrent Learning (RTRL) algorithm. The dataset has samples belonging to 5 different categories 4 of which are different types of attacks. The paper reported 93.82 classification accuracy. Kim et al. used an offline batch training algorithm relying on using an LSTM network trained using BPTT. The paper reported 99% training accuracy [8]. Both papers used the KDD'99 dataset and concluded that while LSTM shows promising results, it cannot detect infrequent attacks in the dataset.

[22] tested the effect of the size of the LSTM network on the detection performance using NSL-KDD. [23] addresses one of the intrusion detection system challenges, which is to achieve a low false alarm rate with new unseen threats using LSTM classification. The authors built a model using different RNN models to identify seen and unseen threats. They tested their model on unseen network attacks. [22] also compared network models containing multiple LSTM layers and concluded that adding more layers does not improve the LSTM classification performance.

In addition to these supervised methods, several deep unsupervised methods have been tested for intrusion detection. [24] proposed a deep belief network model in which an unsupervised greedy learning algorithm was used to learn similarity representations

of the data. KDD'99 was used for evaluation and the authors concluded that the network performs better than SVM in terms of accuracy.

[25] tested stacked deep autoencoders on KDD'99 and showed that a model containing 4 stacked autoencoders can detect the network attacks in KDD'99 dataset with 94.71% accuracy.

[26] utilized LSTM autoencoders for intrusion detection with their proposed autoencoder framework for both fixed and variable length data sequences. They developed an online sequential unsupervised dataset for network intrusion detection using LSTM-autoencoders on ISCX IDS 2012 dataset [27]. Their experiment carried out 5-fold cross validation to validate the performance of their framework using the dataset. The experiment carried different autoencoders such as LSTM-Autoencoder with Last pooling, LSTM-Autoencoder with Max pooling, LSTM-Autoencoder with mean pooling.

In this thesis, variants of these existing supervised and unsupervised, DNN and LSTM architectures have been implemented in addition to Temporal Convolutional Networks (TCN), which is described in detail in section 3.7. The existing works use different metrics to measure the performances of the algorithms. Some of these works do not measure zero-day attack detection capabilities of the algorithms or detection rates for different types of attacks. In addition to that, these works focus on a single algorithm and do not compare deep learning methods with each other. In this work, the performances of these existing algorithms and their zero-day attack detection capabilities are evaluated. For anomaly-based intrusion detection methods, comparison of these capabilities are as important as other metrics, since detection of unknown attacks is the advantage these methods have over signature-based anomaly detection methods. In addition to that, comparison of these algorithms is an important task since computational complexities of the algorithms are different. The more computationally expensive algorithms should perform considerably better in order for them to be considered useful.

CHAPTER 3

BACKGROUND

3.1. Introduction

This chapter introduces concepts and technology relevant to anomaly-based intrusion detection systems using deep learning. These include:

- Artificial neural networks (ANN) and how to train them
- Recurrent Neural Networks
- Temporal Convolutional Networks
- Autoencoders

3.2. Artificial Neural Networks (ANN)

Most modern deep learning models used today are based on artificial neural networks. Artificial Neural Networks are universal function approximators, which means they are able to approximate any existing function between an input x and output y provided that there are enough hidden units between them. The purpose of training a neural network is to find a function, which can describe the relationship y=f(x) between the sample input and outputs, which can be generalized to data that are not in the sample dataset [14].

Artificial neural networks consist of basic computational units called artificial neurons, which are inspired by biological neurons. An artificial neuron takes a number of inputs to produce an output. The formula used for calculating the output is shown below:

$$y = \sigma(\sum_{i=0}^{n} w_i * x_i + b)$$

The formula used has weights w_i for each input x_i of the neuron, which determines how that input affects the output and a bias value, which is equal to the value of the function when there is no bias. The weights and bias are used to calculate the weighted sum of the inputs [28].

After the weighted sum is calculated an activation function σ is used to calculate the output of the neuron. The activation function is a non-linear function, which allows non-linear transformation of the input.

The non-linearity of the activation function plays a significant role as it allows the network to learn more complex, non-linear tasks. The most common activation functions are sigmoid, hyperbolic tangent(tanh), and Rectified Linear Unit (ReLU) functions. The sigmoid function takes an input and transforms it into a real number in the range [0,1]. The sigmoid function is generally used with neural networks that need to output only positive values (e.g. probability).

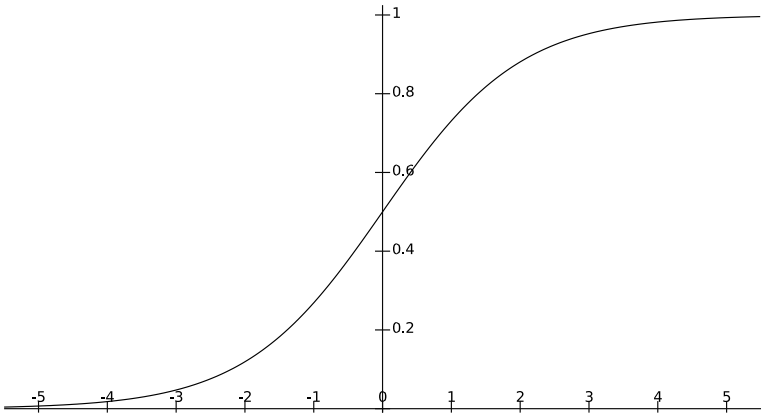

Figure 3.1. Sigmoid Function

Hyperbolic tangent function is similar to sigmoid function, but instead of the [0,1] range, the output of this function is between [-1,1]. RELU function squashes the negative input to zero while outputting the positive input as it is. RELU function is computationally less expensive than the other activation functions. Unlike the

previous activation functions, ReLU does not saturate to -1, 0 or 1 and converges faster.

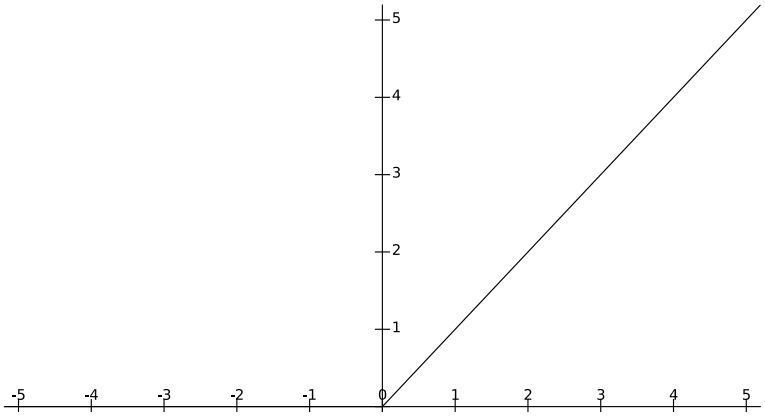

Figure 3.2. RELU activation function

Neural networks consist of artificial neurons that are organized in layers. The network takes the input and processes it layer by layer to produce an output.

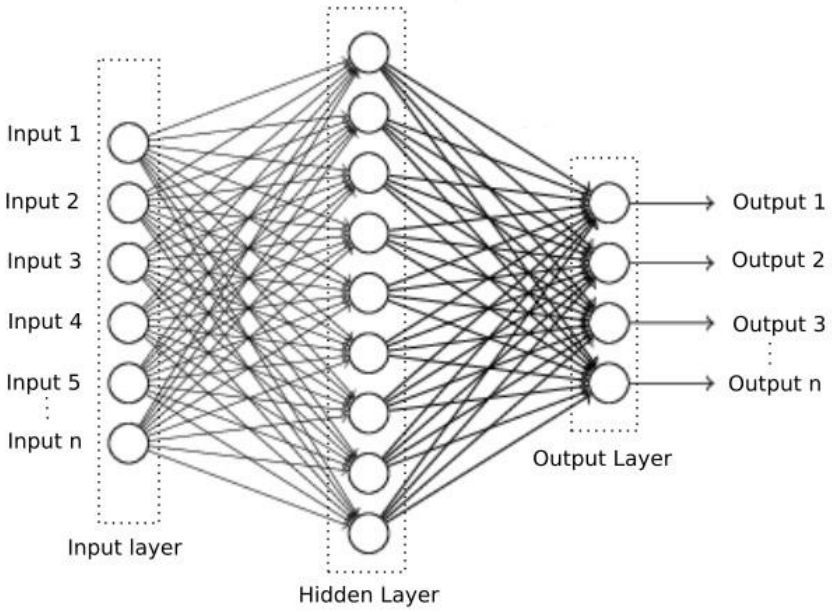

Figure 3.3. Neural Network with Single Hidden Layer

3.3. Feed Forward Neural Networks

Neurons are basic building components of the neural network. In a neural network, artificial neurons are organized in layers, where each node is connected to nodes in the previous layer. The first layer of the network is called the input layer since this layer takes the input of the network as its input. The last layer is the output layer and the output of this layer is also the output of the neural network. The remaining layers are called hidden layers. Each neuron in each layer is connected to the output of each node in the previous layer. Each neuron takes the outputs of the previous layer's neurons and produces an output for the next layer. The flow of information starts from the input layer, passes through hidden layers to arrive at the output layer. At the end, the output of the network is produced [14].

3.4. Training a Neural Network

The goal of the training is to find the optimal weight values for each neuron so that the outputs produced by the model are closest to the actual outputs given in the sample data input.

In order to measure the capability of a neural network to predict the correct output for a given input, a function called *loss function* is used. Loss function measures the difference between the real output and the output estimated by the network. One of the most used loss functions is Mean Squared Error (MSE), which calculates average squared values of the real and estimated output values.

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \text{ , where}$$

y is the actual outputs and $\hat{y}$ is the estimated outputs.

Another common loss function is binary cross entropy (BCE). BCE is used when the output is binary.

$$BCE(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i)(1 - \log(\hat{y}_i)), \text{ where}$$

y is the actual outputs and $\hat{y}$ is the estimated outputs.

The goal of training a neural network is to minimize the selected loss function by adjusting the weight and bias values of the neurons. Training is done using a supervised training algorithm called backpropagation, which takes sample data as input and output pairs.

The training algorithm starts by initializing the weights of the network randomly. After that, two phases, *forward pass* and *backward pass* are repeated until the loss is minimized.

In forward pass phase, the output of the network and the value of the loss function are calculated. Then, in the backward pass phase how to update weights so that the loss is reduced is found by using the gradient descent algorithm and weights are updated by a small amount.

The gradient descent algorithm is used to determine how to update the weights to reduce the loss. *Gradient* is defined as corresponding change in the loss value when a weight parameter is updated. Gradient descent algorithm calculates the gradients of each network parameter with respect to the loss function using chain rule and uses the direction of the gradient to determine how to reduce the loss value.

The chain rule is used to calculate the gradient of the loss function. Chain rule states that, for a forward propagating function $f(x) = A(B(C(x)))$ where A,B and C are functions used to calculate the value of $f(x)$, the derivative of the function with respect to x is:

$$f'(x) = f'(A)\,.\,A'(B)\,.\,B'(C)\,.\,C'(x)$$

The forward propagation function for a neural network is $\hat{y} = f(X) = L(\sigma(WX))$ where L is the cost function, σ is the activation function, W is the weights of the network and X is the input. Thus, the gradient of the loss function is calculated using formula below for each weight w_i in the network:

$$\frac{\partial(Loss)}{\partial w_i} = \frac{\partial L}{\partial \hat{y}}\,\frac{\partial \hat{y}}{\partial \sigma}\,\frac{\partial \sigma}{\partial w_i}$$

After the gradient is calculated, the weights are updated in a small step using the formula below. α is a parameter called *learning rate*, which determines the step size of the updates. Using the calculated gradient, learning rate α, and the old value of the weight w_i, the new weigth value $\grave{w_i}$ is calculated using formula

$$\grave{w_i} = w_i - \alpha \frac{\partial(Loss)}{\partial w_i}$$

The training algorithm decreases the loss incrementally in small steps until it cannot decrease anymore using these two phases until the training is completed.

For large datasets, calculating the gradient for each training sample is time consuming; therefore, an algorithm called stochastic gradient descent (SGD) is used to speed up the process. Instead of calculating the gradient for each sample, SGD splits the input into batches and calculates the gradient descent for all elements in a batch together. SGD along with its variants such as Adam, Adagrad, RMSprop are commonly used algorithms for training [29].

3.5. Recurrent Neural Networks (RNN)

The feed forward neural networks assume that each sample is independent from each other. However, this assumption is not always true. In fields such as natural language processing, time series classification, video etc. there is a temporal dependency between the data points across time. Feedforward neural networks similar to many other machine learning algorithms ignore such dependencies. RNN is an extended version of the previously defined feed forward network to handle these dependencies.

RNN works on sequential data and while processing the sequence retains information about the past elements in the sequence. In order to do this, RNN keeps a hidden state parameter, whose value is determined by the past elements of the sequence.

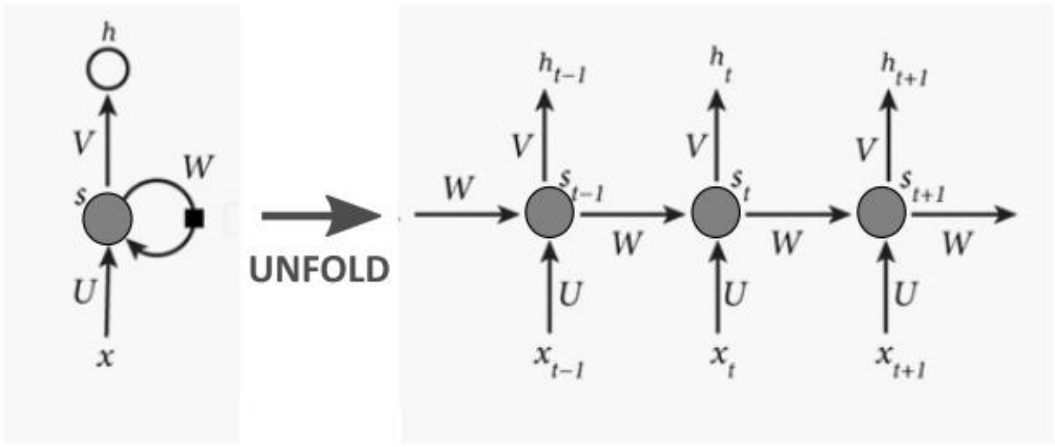

Figure 3.4. Recurrent Neural Network Architecture

While processing the element of sequence at time step t, the network uses the hidden state of time step t-1. The mathematical formula for calculating the output of an RNN is:

$$s_t = \sigma(Ux_t + Ws_{t-1} + b_s)$$

$$h_t = \sigma(Vs_t + b_h)$$

In this formula x_t is the input at time t and s_{t-1} is the hidden state's previous input. The current state and current output of the network is calculated using these values. *U* are the weight parameters, which determine the importance of the current input to the output similar to the feed forward network. W and V are additional weight parameters, which determine the importance of the past data to the output. The training algorithm for recurrent neural networks must calculate the optimal values of these weight parameters in a similar way to the previous network model.

An RNN can be unrolled in temporal axis to convert it to a feed forward neural network. Unrolling can be done by creating a copy of the RNN for each time step. The difference between a normal feed forward neural network and an unrolled recurrent neural network is that the unrolled network has the additional constraint that the weights must be shared across the model. Because of this, recurrent neural networks

can use a variant of the backpropagation algorithm called Backpropagation Through Time (BPTT) [30].

In BPTT, backpropagating the gradients through the whole network is required in order to calculate derivatives of the cost function with respect to each weight of the network via the chain rule. As the depth of the network increases, the gradients can become extremely large or small because of many multiplications required for applying the chain rule. When the gradients become large, the network suffers from the exploding gradient problem in which very large weight updates cause problems such as unstableness of the network and overflow of the weights. On the other hand, too small weights cause the vanishing gradient problem, which result in very small weight updates to stop the network from learning [14] [31]. Because of vanishing and exploding gradients, the RNN can only be trained on sequences shorter than 10-time steps [21].

3.6. Long Short-Term Memory Networks (LSTM)

LSTM is a variant of RNN, which was proposed by [32] to combat the vanishing and exploding gradient problem of RNN. LSTM uses gates to determine whether to store a specific element in the hidden state or forget it, which allows to remember longer dependencies.

LSTM uses gates to determine whether to forget or keep the incoming data in the hidden state. The gates are implemented using sigmoid functions, whose output is in the range between 0 and 1. If the output of the gate is closer to 0, the LSTM does not allow the input to pass through [33].

LSTM architecture consists of four main components: forget gate, input gate, output gate and memory cell. The forget gate decides whether to forget the current input or not based on the current input and hidden state values, while the input gate decides which part of the input to keep in the hidden state.

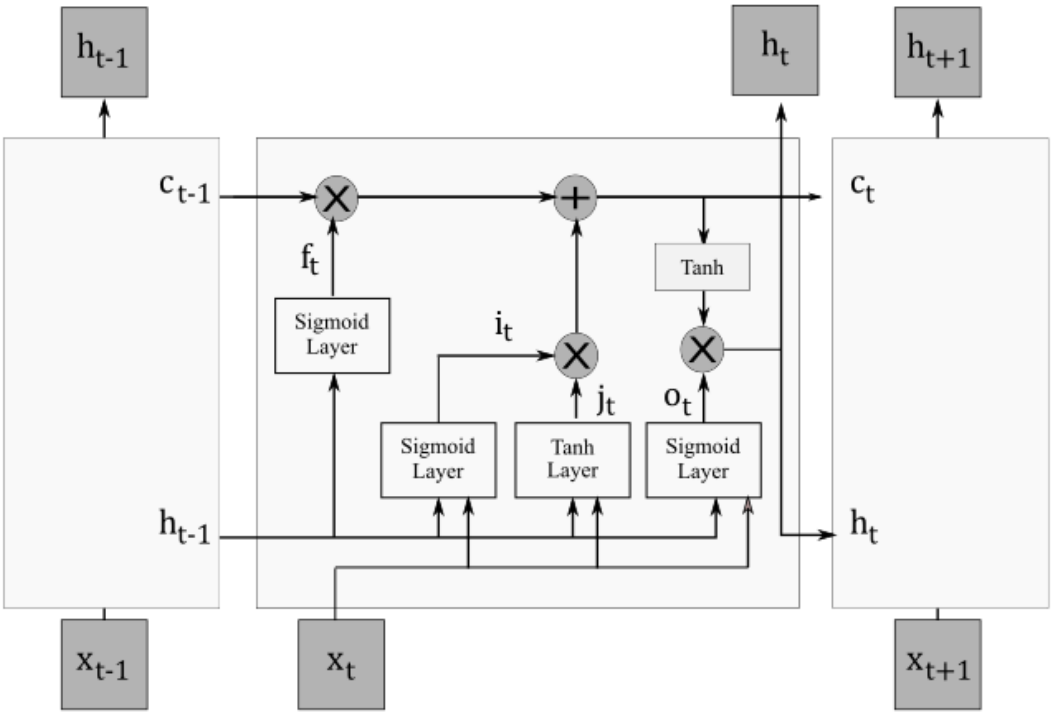

Figure 3.5. LSTM Architecture [34]

f_t is the output of the forget gate and W_f, U_f, b_f are weights and bias parameters of the forget gate. i_t is the output of the forget gate and W_i, U_i and b_i are input gate weights and bias. These weight and bias parameters are parameters that are optimized during training.

$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$$

The output of input and forget gates, along with the current value of the input are used to calculate the value kept at the memory of the cell c_t. After that, the output and hidden state are calculated using these values. $\odot$ is the element-wise vector product.

$$j_t = tanh(W_j x_t + U_j h_{t-1} + b_j)$$

$$c_t = (f_t \odot c_{t-1}) + (i_t \odot j_t)$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

In RNN architectures, in order to calculate the output for an input at time *t*, the output of the previous steps must be calculated beforehand. Therefore, the training process for RNNs cannot be parallelized. In addition to that, the architecture of the LSTM cells is more complex than feed forward neurons. Thus, training RNN architectures require a lot more time and computational power.

3.7. Temporal Convolutional Networks (TCN)

Since RNNs are time and memory intensive, several architectures were developed to replace them. One such solution is Temporal Convolutional Networks. TCN is an architecture which can take an input sequence of any length to produce an output of same length similar to RNN [35], [36].

Convolutional Neural Networks are a specific type of feed-forward networks in which the input is arranged in a grid-like structure. A CNN consists of convolutional layers and pooling layers. A convolutional layer consists of a set of filters. In forward pass, the filter is moved across the grid structure of the input and the dot product between the input value and filter are calculated to produce the output. The size of the filter is smaller than the size of the output, which allows extraction of spatial features. Instead of weights for each input element, the filters of convolutional layers are the network parameters which are optimized via training. Therefore, CNN is computationally less expensive than feed forward networks. Standard CNN does not work on sequential data. TCN is an architecture, which uses convolutional layers to process sequences.

In order to do sequence processing, TCN uses convolutional layers followed by zero-padding layers. The size of the output of a convolutional layer is smaller than the size of input. Thus, zero-padding layers are used to keep the size of the input and output the same and the output size does not change after convolutions.

Another modification of TCN is that, in the convolutional layer, the output at time *t* is calculated by only using elements from time t and earlier in the previous layer. This

type of convolution is called “causal convolution”. Causal convolutions allow sequence classification using only the past and current information, since it prevents future data from affecting the results.

TCN also uses dilated convolutions to be able to look back at data from further back. With only causal convolutions, the TCN can only look back at a history at depth linear to the network depth. Dilated convolutions allow the network to keep history at depth exponential to network depth. Figure 3.6 shows visualizations of causal convolutions with and without dilations.

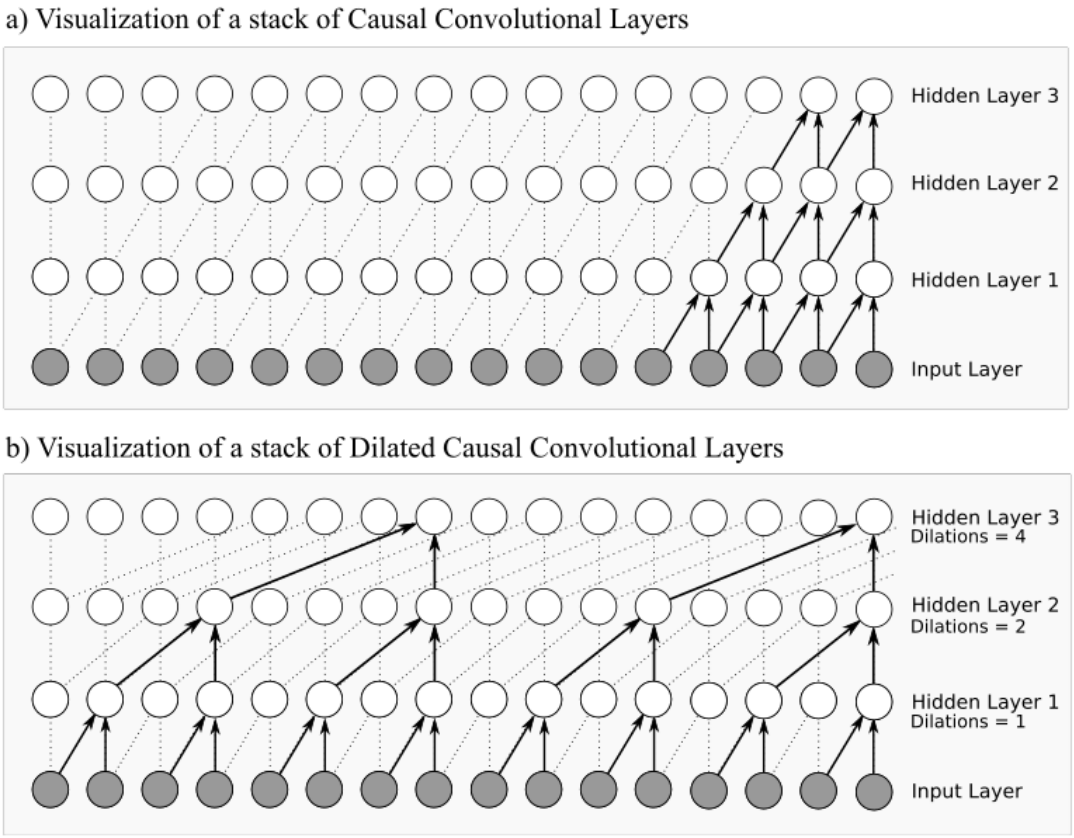

Figure 3.6. Causal vs Dilated Causal Convolutional Layers [37]

To create a TCN network, residual blocks whose diagram are shown in Figure 3.7 are used. Residual blocks are stacked on top of each other to create the TCN.

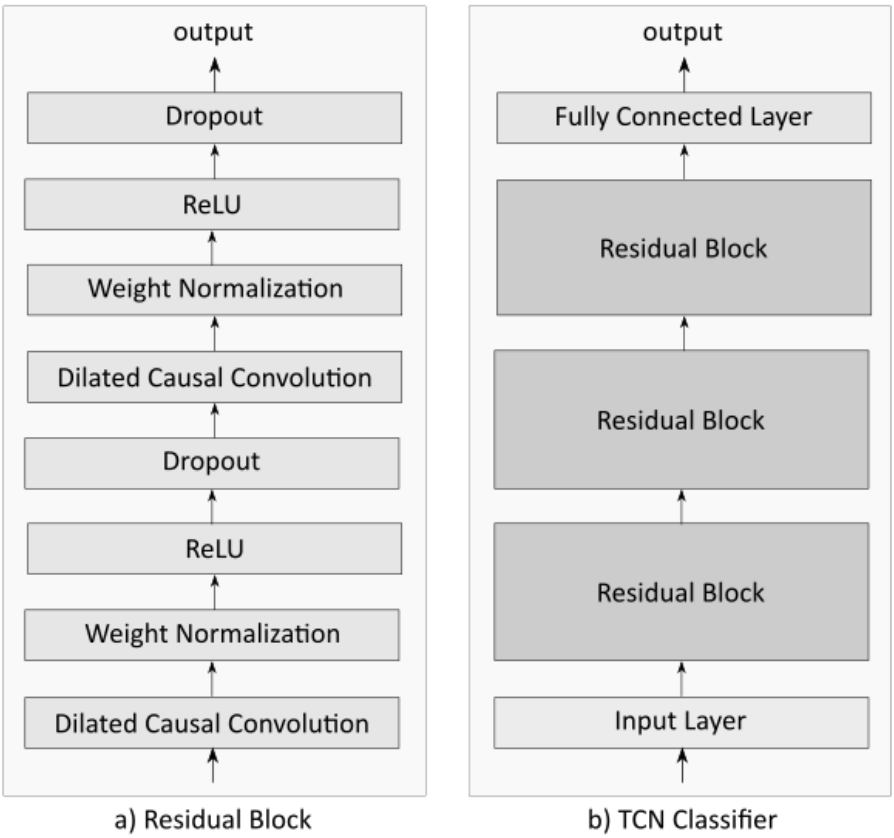

Figure 3.7. Residual Block and TCN Classifier

TCN share filters across layers and the depth of the path backpropagation algorithm depends only on the depth of the network. Because of this, TCN have low memory requirements in comparison to recurrent neural networks, which require memory to store partial results for all gates of its recurrent cells. In addition to that, convolutions of TCN network can be done in parallel while recurrent architectures cannot be parallelized. [35] compared LSTM and TCN on 11 RNN problems and showed that TCN works at least as well as LSTM in every task.

3.8. Autoencoder

Autoencoder is a specific type of neural network that is trained to copy its own input as output with the goal of creating a representation of the input. An autoencoder consists of two parts:

- An encoder that maps the input x using h=f(x) and
- A decoder that reconstructs the input from the output of encoder h.

The goal of training an autoencoder is to find the optimal parameters, which allows the model to output its input as best as it can.

Autoencoders can be used to extract important characteristics of the input dataset. One way to learn useful features from a dataset is to use an autoencoder, which has a smaller dimension in comparison to x. This forces the encoder to extract defining characteristics of input so that the decoder can reconstruct the input using these. If the output space of the encoder is larger than the input size, then the encoder memorizes the input without learning. This type of autoencoder is called an undercomplete autoencoder.

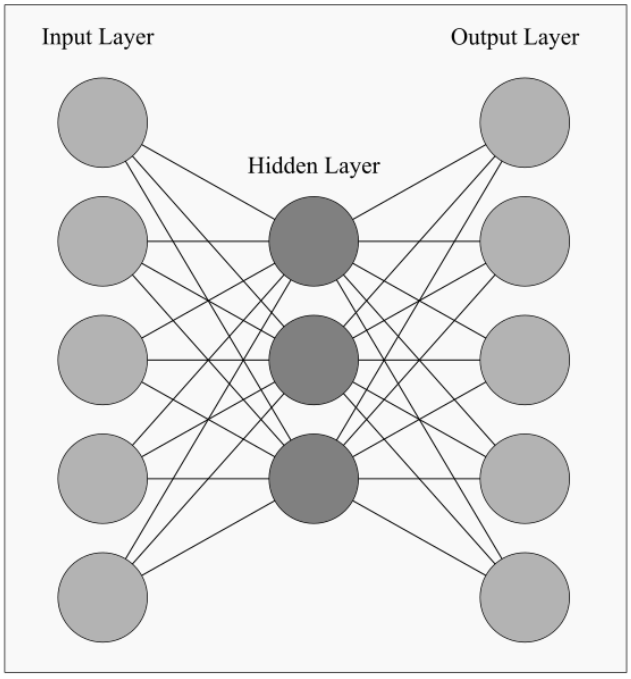

Figure 3.8. Autoencoder

In autoencoders, both feed-forward and recurrent architectures can be used as encoders/decoders. An autoencoder can be used to extract features from both individual inputs and sequences.

3.9. Neural Network Parameters

Neural networks have two types of parameters: trainable parameters and non-trainable parameters. Trainable parameters of the network are parameters whose values are learned during training. These parameters include the weight and bias parameters of the neurons.

Non-trainable parameters of the network are called hyperparameters. These parameters are decided before the training is started and includes parameters such as number of hidden layers and hidden neurons in the neural network, learning rate, and optimization algorithm. The values of these parameters must be tuned through trial and error.

3.10. Evaluation Metrics

Performance of a neural network is evaluated using its confusion matrix. For anomaly detection tasks, in general the data is labelled as negative or positive depending on whether given input contains an anomaly or not. If a sample contains an anomaly, it is labelled positive and if a sample is not anomolous, it is labelled negative. To represent a dataset labelled using these values, binary confusion matrices are used.

A binary confusion matrix is a matrix consisting of the values below:

- True Positive (TP): Actual value is positive; model correctly predicts positive
- False Negative (FN): Actual value is positive; model incorrectly predicts negative
- False Positive (FP): Actual value is negative; model incorrectly predicts positive
- True Negative (TN): Actual value is negative; model correctly predicts negative

These values are used to calculate metrics which measure the performances of anomaly detection algorithms.

Metrics calculated using the values are:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$f1\ score = \frac{2 * Precision * Recall}{Precision + Recall}$$

CHAPTER 4

DEEP LEARNING BASED INTRUSION DETECTION

4.1. Introduction

This section describes the KDD'99 intrusion detection dataset used in this thesis and how the deep learning techniques described in the previous chapters are used on the KDD'99 dataset to detect malicious activities.

4.2. KDD'99 Dataset

KDD'99 dataset is a fully labeled intrusion detection dataset created in 1999 for The Third International Knowledge Discovery and Data Mining Tools Competition. The data was collected from simulation of the USA Air Force network. During the simulation, 7 weeks' worth of TCP dumps were recorded for training and 2 weeks of testing data were recorded for testing [38].

The dataset was created by extraction of 41 features from the collected TCP dumps. The extracted features can be classified into 4 categories: Basic Features, Content Features, Time-based Traffic Features, Host-based Traffic Features.

Basic Features: Basic features are the features extracted from the packet headers without examining the packet contents. Basic features consist of the following parameters:

- Duration: The time between the start and end of the connection. This feature is a continuous feature.
- Protocol: The protocol used for communication. The possible values for this categorical field are "tcp", "udp" and "icmp".
- Service: The service field specifies services such as telnet, ftp, http etc. The dataset includes about 70 service categories.

- Flag: The status flags of the connection. e.g. S0 flag, which means a connection is established, REJ which means a connection is rejected.
- Source Bytes and Destination Bytes: Continuous data containing information about the number of bytes sent from source to destination.

Content Features: Content features are features extracted from the payloads of the packets using domain knowledge. This category includes features such as number of failed login attempts, number of file reads made, whether the connection requested actions using admin privileges etc.

Time-based Traffic Features: The time-based features are collected using 2 second windows to capture temporal features such as number of connections to the same and different hosts in the last 2 seconds, the error rates of the server, etc.

Host-based Traffic Features: Host-based features are extracted using windows containing the last 100 packets. These features are helpful in machine learning for detection of attacks spanning longer than 2 seconds.

The training dataset contains 24 different attacks, the testing dataset also contains these 24 attacks but in addition to these, there exist 14 extra attacks in the testing dataset. The addition of these attacks aimed to test the abilities of the intrusion detection systems on unseen data [39].

The attacks can be classified into 4 different categories. These 4 categories are:

- Denial-of-Service (DoS): Attacks whose purpose is to deny access to a network by overflowing the network with malicious packets and consuming all resources.
- Probe: Attacks whose purpose is to collect information about the network. Probe attacks use methods such as scanning the ports or IP addresses to detect vulnerabilities of the victim.

- Remote to Locale (R2L): R2L attacks are attacks in which the attacker can send packets to the target over the network but does not have login rights. The goal of the attack is to exploit a weakness to gain user access to the machine.
- User to Root (U2R) Attacks: U2R attacks are attacks such that the attacker has normal user access to the target but tries to gain root access. U2R attacks use methods such as password sniffing to gain access.

Table 4.1 shows the different attacks in each category where the attacks that only exist in the test set are written in bold.

Table 4.1. *Attacks in each attack category in KDD'99 Dataset*

Category	*Attacks*
DOS	back, neptune, smurf, teardrop, land, pod, **apache2, mailbomb, processtable, udpstorm**
PROBE	satan, portsweep, ipsweep, nmap, **mscan, saint**
R2L	warezmaster, warezclient, ftp_write, guess_passwd, imap, multihop, phf, spy, **sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm, httptunnel**
U2R	rootkit, buffer_overflow, loadmodule, perl, **ps, sqlattack, xterm**

There are some problems with the KDD'99 dataset. The first problem is that it is old; therefore, it does not contain many of the services used today.

The second problem of the dataset is that it is imbalanced. The training dataset contains 494019 samples, which are distributed as 19.69% normal samples, 79.24% DoS attacks, 0.83% Probe attacks, 0.23% R2L attacks and 0.01% R2L attacks.

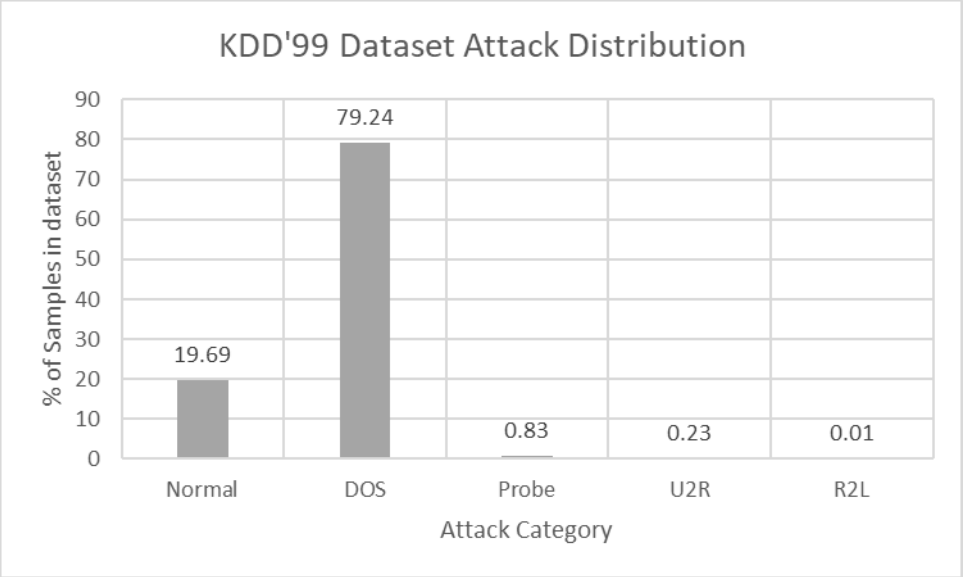

Figure 4.1. KDD'99 Attack Distribution Graph

In the dataset, less than 1% of the attacks are R2L and U2R attacks, which makes training and testing intrusion detection methods for these types of attacks harder. Because of this, many of the traditional intrusion detection systems, especially those that use classification-based algorithms are unable to detect these intrusions with a better ratio than random guessing.

The third problem of the dataset is the noise of extracted features. There are samples in the dataset, which contain the exact same or very similar features with different labels. This noise especially affects the detection rates of the U2R attacks since their features are most similar to the normal samples.

Other problems regarding the simulation used to collect data also exist. For example, in 2003, [40] discovered that all the malicious packets had a TTL of 126 or 253 and almost all the normal packets had a TTL of 127 or 254. Fortunately, this property was not used in the extracted features.

Despite the problems, the KDD'99 dataset is the most used intrusion detection dataset by far. It is used in 64% of the intrusion detection papers, which can be considered as the benchmark for comparing different intrusion detection algorithms [13]. It includes

9 weeks of data in total, which includes many different connection types. The distributions of the attacks are different in the training and testing set and there exist additional attacks in the testing set. These differences allow better testing of intrusion detection algorithms. Especially the additional attacks allow testing of intrusion detection systems on whether they can detect unknown attacks or not. Because of these, the KDD'99 dataset is selected to be used in this thesis for testing various deep learning algorithms.

4.3. Deep Learning Based Intrusion Detection Algorithms

The deep learning-based intrusion detection algorithms include supervised binary classifiers and unsupervised autoencoders. Binary classifiers use all of the training data to learn to separate data into normal and attack classes. Autoencoder-based unsupervised methods use only non-anomalous training data to learn the normal of the system and try to find samples, which deviate from the normal.

In this thesis, standard Fully Connected Network and sequential LSTM and TCN networks are used as both classifiers and autoencoders. The next sections describe how these architectures are used as a classifier and autoencoder.

4.4. Binary Classification Using Deep Learning Techniques

Given a dataset containing normal and anomalous samples, binary classification can be used for anomaly detection in a straightforward manner. After preprocessing the data, different hyperparameters for the neural network must be tested to find the optimal neural network. Some of the hyperparameters of the feed forward neural network are explained below.

Learning Rate and Optimizer Algorithm: There are several optimization algorithms which are based on SGD. [41] developed a unit testing framework to compare these algorithms and concluded that different optimization algorithms struggle on different tasks and there is no clear best among them. Schaul also stated that *Adam optimizer* is less prone to hyper-parameter tuning. Adam is an adaptive

learning method which calculates a learning weight value for each individual network parameter by using gradient estimations. Generally, Adam is faster than SGD in training the model, however, [42] showed that, despite training taking longer, SGD achieves better accuracy rates than Adam for some tasks. Therefore, Adam and SGD are selected as potential optimization algorithms.

Network model: The number of layers and the number of neurons in each hidden layer need to be decided. There is no certain method to decide this, other than trial and error. But there are some general guidelines thanks to the previous works in the deep learning field.

The more complex the relationship between input and output data, the larger networks are required. If the network is not large enough for the complexity of the problem, then the network would underfit and cannot learn the complex relationship between them. If the network is too large, the network can overfit and simply memorize the sample data and would not learn anything else [43]. The amount of training data is also another factor, which sets a maximum limit for the network model, if data is smaller than the total number of networks parameters, the parameters cannot be trained well enough [44].

The main problem of classifier-based anomaly-based detection techniques for this task is the dataset imbalance. Including the KDD'99 dataset, anomaly detection datasets generally suffer from dataset imbalance, since anomalous data appears much less frequently than normal data. This makes the deep learning model biased against anomalous data, since misclassifying normal data affects the cost function more. In the intrusion detection case, the collected datasets usually have sufficient amounts of data for some types of attacks such as DoS, but other types of attacks usually do not have enough samples.

Because of the imbalance problem in the KDD'99 dataset, it is very hard for anomaly-based detection systems to detect R2L and U2R classes while DoS and Probe attacks

are easier to detect. In fact, most machine learning algorithms cannot detect R2L and U2R attacks with better accuracy than random guessing [21].

Dataset imbalance is one of the biggest challenges of deep neural networks. There are some methods proposed to fight with imbalance. These methods can be classified into two categories: Sampling-based and Algorithm-based. Sampling-based methods can be further divided into two categories: Undersampling and oversampling.

In undersampling-based methods, only a subset of the majority class is used in classification to make the dataset balanced. The disadvantage of this method is the information loss regarding the majority classes. However, since KDD'99 dataset is known for its duplicate and redundant records, this method might help detect the minority classes while not losing information.

In algorithm-based methods, the algorithm is modified to handle the imbalance. One example of such methods is cost-sensitive loss functions. Usually, the loss functions used in deep learning do not differentiate between the misclassification of different classes. However, modifying the loss function to punish misclassification of the minority classes more helps classification of minority classes. One of the most popular methods used for this purpose is to add weights to the loss function disproportional to the size of the samples of each class. These two methods will be used to experiment on the KDD'99 dataset to solve the imbalance problem.

Experiments were run with different parameters such as learning rate, number of layers in the network and number of neurons in each layer and regularization parameters. The process of finding good parameters is empirical, iterative and requires several round trips.

4.5. Binary Classification Using Sequential Models

Feed forward neural networks assume each sample is independent from each other, however, this is not always the case for intrusion detection data. For example, DoS attacks, which aim to overflow the network by sending multiple packets at once, and

probe attacks, which search for weaknesses in the target, are sequential attacks by nature. The network packets carrying these attacks can look like non-anomalous data, but the frequency of the packets might indicate that an attack is happening.

In addition to that, the multi-step attacks which consist of multiple independent attacks following each other can be detected from sequential data. For example, in KDD'99 data *probe* attacks are used for detecting weaknesses, U2R attacks are used for gaining user access to the target and R2L attacks are used for gaining admin access from a computer in which the attacker has user access. In that case these samples are clearly not independent from each other.

LSTM and TCN are architectures used when there are such dependencies between the samples. They work on sequences and remember past samples in the sequence while classifying each sample.

LSTM can handle sequences of 500 samples using BPTT without vanishing and exploding gradient problem. Exploding gradients can be solved by clipping the gradient values above a threshold, but vanishing gradients are a challenge for LSTM networks. In addition to that, it is computationally expensive to run BPPT for long sequences [45].

The KDD'99 training dataset is a 494019-time step sequence. Therefore, LSTM cannot be used on the dataset without segmenting the data into smaller sequences. In order to apply LSTM to long time series, Sutskever proposed a method called "truncated BPTT" (tBPTT) which "processes the sequence one timestep at a time, and every k1 timesteps, it runs BPTT for k2 timesteps…". Since the gradient is only calculated for k2 steps, this solves the vanishing gradient problem while still allowing LSTM to learn the past. tBPTT solves the vanishing gradient problem while at the same time reducing the cost of a single parameter update. Sutskever shows that tBPTT works by giving an example in which a sequence of length 1000 is divided into sequences of length 20. In this example, the algorithm was able to learn temporal dependencies of the sequence without facing vanishing gradient problem [46].

The TensorFlow library implements a specific version of tBPTT, which is called “epoch wise truncated backpropagation”. In this variant of tBPTT, k1 is always equal to k2. The sequence is processed a fixed number of timesteps and then BPTT runs for the same fixed number of steps. Thus, what the algorithm does is essentially divide the sequence into fixed sized smaller sequences and process each sequence by using the output and hidden state parameters of the previous sequence. The researches comparing Sutskever’s tBPTT and TensorFlow’s tBPTT show that the performance of TensorFlow tBPTT with step size k is equal to the performance of Sutskever’s tBPTT where k1 is 2*k and k2 is k [30], [47]. Feed forward autoencoders are trained on each normal sample to reconstruct the normal samples. Similar to classification models, the network parameters must be configured for optimal performance.

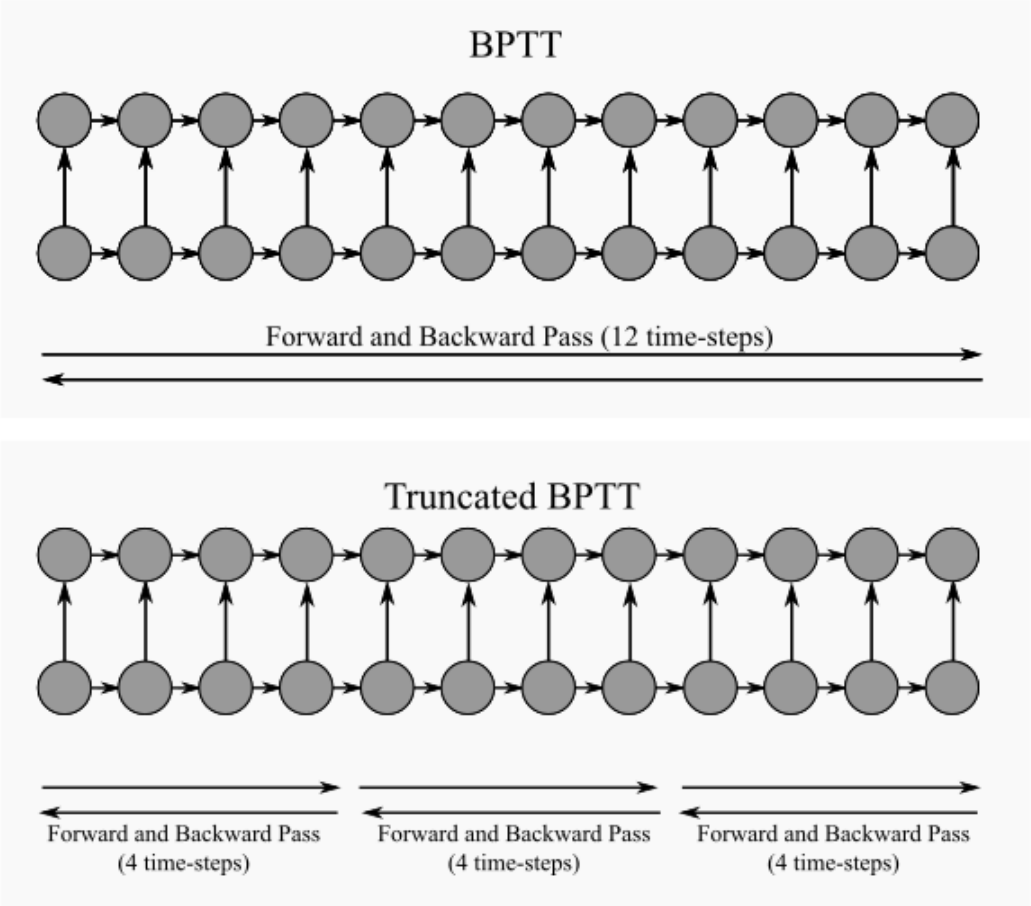

Figure 4.2. BPTT vs Epoch-Wise Truncated BPTT

The main problem of tBPTT is that it is sequential, therefore before processing a batch, the output and hidden state of the previous batch need to be known. Therefore, the process cannot be parallelized. Because of this and the more complex structure of the cells, LSTM is much slower than the feed forward networks while also being computationally more expensive. Therefore, LSTM is at a disadvantage in comparison to feed forward neural networks, especially if the model is used for real time intrusion detection.

In this thesis, Feed forward, LSTM and TCN classifiers have been implemented for intrusion detection.

4.6. Unsupervised Method: Autoencoders

To use an autoencoder for anomaly detection, the model first must be trained using only the non-anomalous samples of the training set so that the model learns what the normal baseline for the dataset is. Then, a reconstruction error threshold must be selected above which the samples are considered anomalous. In the literature there are different methods used for selecting the threshold. These methods generally use two validation datasets consisting of only normal data and mixed data respectively. The thresholds are selected using the reconstruction errors of the normal validation data generally using statistical properties of it such as mean, standard deviation, etc. Selected thresholds are then tested on the second validation dataset.

For LSTM and TCN autoencoders, the reconstruction is done not for individual samples, but for sequences. For these autoencoders, fixed size sequences are extracted from all datasets using a sliding window. The label of each window is decided by whether the window has any anomalous samples or not. It is expected that the autoencoder should not be able to reconstruct a window sequence if anomalous data exists regardless of the location of the anomaly. While calculating the reconstruction errors, the whole sequence is used.

The proposed LSTM autoencoder model does not use tBPTT and treats the fixed-window sized sequences as if they are independent of each other. This approach was first used by [48] to detect anomalies in multivariate sensor data. This work empirically showed that despite the sequence is divided into smaller sequences independent from each other, the model was still able to extract sequential information from the dataset. Later on, [49] used a similar algorithm to detect anomalies in seasonal KPIs (key performance indicator). Despite using fixed sized windows, which hold fewer timesteps than a season's worth of data, the autoencoder was able to extract seasonal patterns from the dataset. [49] proved that the autoencoders are able to extract patterns which span larger than the fixed size window length formally and called this

“time gradient effect”. Using this approach, fixed-size sequences can be processed in parallel, which speeds up the training time considerably.

After the model is trained on normal data, an anomaly score threshold is calculated for the whole sequence. If the score of a sequence is above a decided threshold, the sequence is considered an anomaly. The hyper parameters need to be tuned for this method, including window size, number of LSTM layers in encoder and decoder, sizes of LSTM layers, the anomaly score threshold in addition to regular network parameters such as regularization, learning rate and so on. Similarly, TCN hyperparameters also must be tuned.

A disadvantage of this method is that it assigns a label to the whole sequence, not each sample. Thus, it is not possible to determine the exact samples that cause the anomaly. However, since a sliding window is used to create smaller sequences, it is possible to pinpoint where the anomaly is in the sequence by looking at where the first anomalous record is.

CHAPTER 5

EXPERIMENTS

5.1. Introduction

This section contains the performance results of the anomaly detection methods described in the previous chapters. In the experiments, at first, optimal parameters for both supervised and unsupervised methods are found. Then, the best performing models are compared with each other. Lastly, the results are compared with SVM, which is one of the best performing machine learning algorithms in the intrusion detection field.

5.2. Data Pre-processing

KDD'99 contains 38 continuous or binary and 3 categorical features. The categorical features are: "Protocol type" which has 3 categories (tcp, udp, icmp), "service" which has 70 categories (including ftp, telnet, http etc.) and "Flag" including 11 categories. The neural networks require numerical input, therefore the categorical features are converted to numerical values using one-hot-encoding.

One of the features in the dataset, num_outbound_cmds, only has a single value in both training and testing datasets, therefore this feature is dropped from the datasets.

The numerical features are normalized using min-max scaling [50] so that each feature ranges in the interval [0,1].

The outputs of the samples are converted to binary, where 0 represents a non-anomalous sample and 1 represents a malicious sample.

In addition to that, for training of feed forward neural networks, the duplicate records are dropped from the training dataset. Feed forward neural networks assume every

sample is independent from each other and does not use the dependencies between the samples while LSTM and TCN models can use the relationship between duplicate data to extract information.

5.2.1. Classification Experiments

Hyperparameter tuning is the task of finding the optimal hyperparameters for the neural networks. There are no deterministic methods to decide the hyperparameters for a particular algorithm used on a particular dataset, although there are some rules of thumb, which provide a starting point. Because of this, experiments with different hyperparameters must be ran to find the optimal neural network model.

For each experiment, the tests were run up to 250 epochs. However, an early stopping mechanism was adopted to stop the training if the validation loss started increasing significantly. Additionally, the model was recorded whenever the validation loss hit a new minimum value. For regularization purposes Batch Normalization and Dropout are used.

5.2.1.1. Hyperparameter Tuning for Fully Connected Networks

For the feed forward network all hidden layers are fully connected layers. For this type of network, the hyperparameters to be tuned are learning rate, optimizer algorithm and the network models for each network architecture.

The rule of thumb for selecting learning rate is that learning rate must be a value between 0.00001 and 1 [51]. Therefore, experiments with learning rate values in this range are run for finding the optimal learning rate. In addition to that, experiments using optimizer algorithms Adam and SGD are also tested.

The number of hidden units is calculated using the formula below:

$$N_h = N_s(\alpha * (N_i + N_o))$$

N_i = number of input neurons.
N_o = number of output neurons.

N_s = number of samples in training dataset.
α = an arbitrary scaling factor, usually 2-10.

Starting from this point, the best architecture is searched following an iterative process and an optimal solution is found. Experiments are run with number of layers ranging between 1 to 5. Also, different number of neurons for each hidden layer, in the range 16 and 512 are tested. ReLU activation is used in all tests. Other hyperparameters that are tuned are learning rate and the dropout rates between the layers.

Table 5.1. *Tested Hyperparameters for FCN Classifier*

Hyperparameter	*Tested Values*
Learning Rate	0.00001, 0.0001, 0.001, 0.1, 0.5
Number of Hidden Layers	1-5
Hidden Neuron Counts in Each Layer	2^3 – 2^8
Optimizer	Adam, SGD
Dropout Rates Between Each Layer	0.1, 0.3, 0.8

The learning rate experiments showed that the network performs better when learning rate is in the range [0.001, 0.0001]. When learning rate is 0.00001, the updates to the network are so small that the early exit mechanism stops training since validation accuracy does not improve. When learning rate is larger than 0.1, the networks cannot learn from the network at all. With each update to the weights the loss and validation accuracy increases and decreases dramatically and eventually the network settles on predicting the same value for each sample.

The best fully connected classifier model architecture is shown in Figure 5.1. The tests showed that there are multiple models which performed as well as the model in Figure 5.1.; however, this model is selected as the best model, because it was the smallest model amongst the best architectures. Smaller architectures use less resources and are faster to train. Also, they tend to generalize better [14], which is why the smallest architecture is selected.

The experiments showed that architectures with fewer number of hidden units than the selected architecture were not able to learn the dataset completely and underfit the data. Since the number of DoS samples in the network were more than the number of samples from other attack categories and DoS attacks have the most unique features compared to other attacks, the neural networks were able to learn these attacks but not the others. The architecture with the lowest accuracy rate had 90.1% prediction accuracy. The prediction accuracy of the model in Figure 5.1. is 93.5% and this is the highest prediction accuracy obtained using fully connected layers.

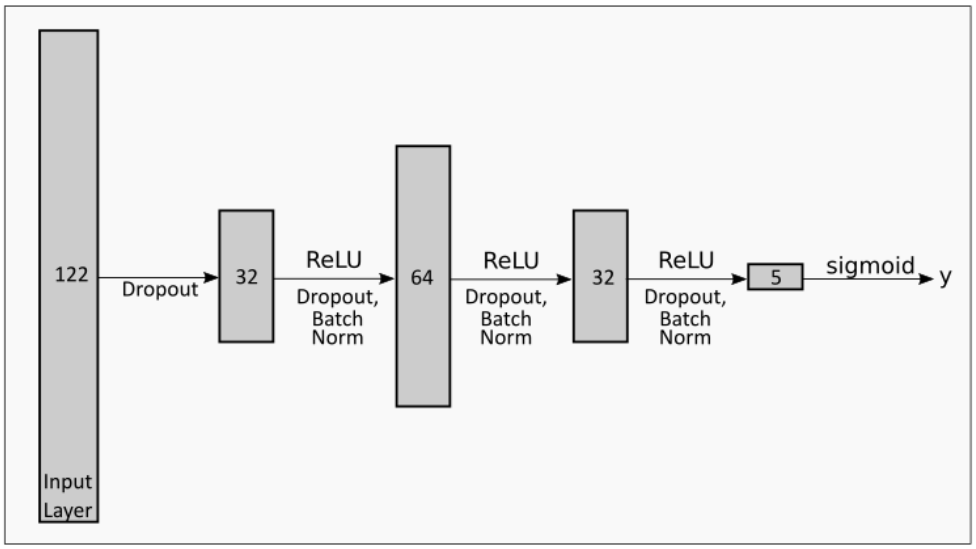

Figure 5.1. FCN Classifier Model

5.2.1.2. Hyperparameter Tuning for LSTM Networks

For LSTM models, the same hyperparameters with the fully connected models must be tuned. However, in addition to these, LSTM models have additional parameters. The first additional parameters are the number of hidden LSTM layers and size of each LSTM layer. For LSTM, the hidden layers usually consist of one or more LSTM layers followed by fully connected layers. Thus, configurations with LSTM layers and fully connected layers are experimented with to find the best solution. In addition, the window size for truncated backpropagation must be selected using the experiments.

Table 5.2. *Tested Hyperparameters for LSTM classifier*

Hyperparameter	*Tested Values*
Learning Rate	0.00001, 0.0001, 0.001, 0.1
Number of Hidden LSTM Layers	1-4
Hidden Neuron Counts in LSTM Layers	32-256
Number of Hidden Connected Layers	1-4
Hidden Neuron in Fully Connected Layers	4, 20, 32, 64, 96, 128
Optimizer	Adam, SGD
Dropout Rates Between Each Layer	0.1, 0.3, 0.8
Time steps for tBPTT	32-512

For the single layer LSTM, different configurations for LSTM hidden neurons and number of timesteps are tested. The experiments were performed using an LSTM layer, which has 32, 64, 128 or 256 hidden units, followed by two fully connected layers with 20 and 1 hidden units each. Gradient clipping is used to limit the gradients within the [-1, 1] range. After the best window size and LSTM hidden neurons are selected, further experiments are conducted to improve the performance.

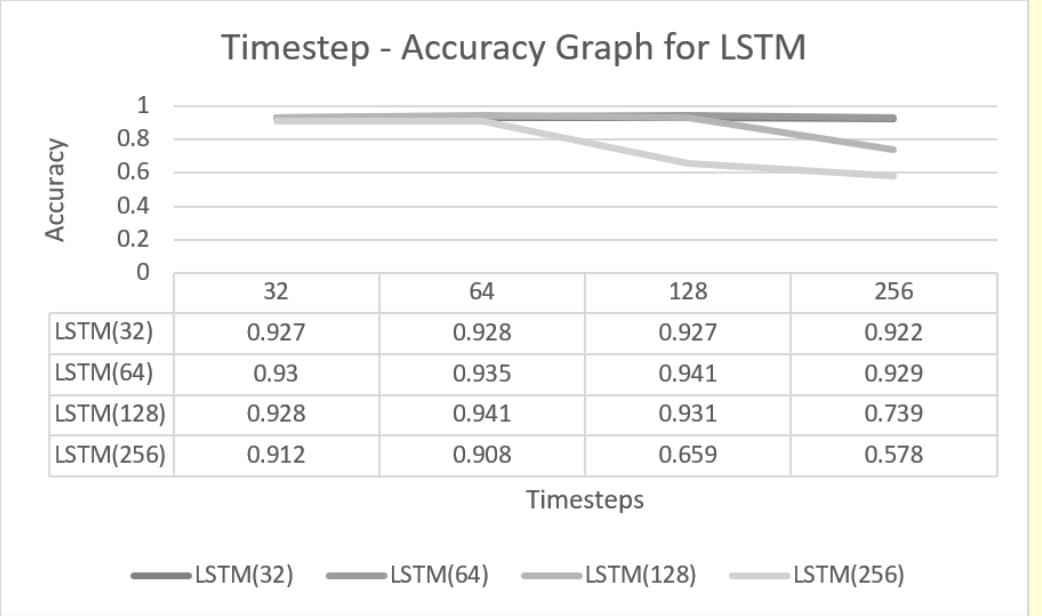

Figure 5.2. Timestep – Accuracy Graph For LSTM

The experiments show that the single layer LSTM network performs the best when the window size for tBPTT is 64 or 128. For longer time steps, the results start to worsen. The selected sequence length is equal to the number of samples processed before an update is made and in the update all samples in that sequence are considered. When the sequence is long, instances of samples, which appear in the dataset rarely, are processed with samples that appear frequently. Therefore, infrequent samples are ignored, and the results are affected negatively.

After selection of timesteps, experiments with additional fully connected layers are conducted. The experiments showed that single LSTM layer followed by 2 hidden layers with 32 and 20 neurons respectively performs very well.

Table 5.3. *Best Performing LSTM networks*

Network	*Time step*	*Acc*	*f-1*	*Normal*	*DOS*	*Probe*	*R2L*	*U2R*
LSTM (64)	128	0.941	0.962	0.995	0.991	0.748	0.093	0.200
LSTM (128)	64	0.940	0.968	0.964	0.999	0.749	0.093	0.157
LSTM (128)	128	0.932	0.955	0.975	0.986	0.788	0.033	0.142

The addition of new LSTM layers did improve the training loss; however, test loss did not reflect this. Therefore, a single layer LSTM is chosen as the final configuration.

5.2.1.3. Hyperparameter Tuning for TCN Networks

For TCN networks, in addition to hyperparameters optimized for fully connected networks parameters regarding the convolutional layers must be optimized. These parameters include the number of convolutional layers, number of filters and kernel size in the convolutional layers. In addition to that, window sizes are decided in an iterative manner. The tested parameters and optimal values are displayed in Table 5.4 below.

Table 5.4. *TCN Parameters*

Hyperparameter	*Tested Values*	*Optimal Values*
Learning Rate	0.00001-0.1	0.0001 – 0.001
Number of filters in Conv layers	16, 32, 64	64
Kernel size for Conv layer	2, 4, 8, 16	4
Number of residual blocks to use	1, 2	1
Optimizer	Adam, SGD	Adam
Dropout Rates Between Each Layer	0.1, 0.3, 0.5	0.1
Activation Function for Conv Layers	Linear, ReLU	ReLU
Sequence Length	32-256	64/128

The size of dilations for each convolutional layer is set by setting the dilation size of the i^{th} convolutional layer to 2^{i-1}.

The optimal learning rate for TCN is learning rates between 0.0001 – 0.001 and is similar to LSTM networks. The experiments showed that the optimal number of filters in the convolutional layers is 64, while the optimal kernel size for the convolutional layers is 4.

5.3. Classification Experiment Results

The results showed that classification models, despite having good performances based on their accuracy and f1-score performances, are unable to detect R2L and U2R attacks. Classification methods suffer from dataset imbalance and cannot detect rare attack types in the R2L and U2R categories since less than 1% of all samples belong to data in this class. Therefore, methods to fight imbalance were applied to the classification methods.

The first method is to apply weights to each sample. Initially weights were selected for each of the 23 unique attacks inversely proportional to the number of samples for each attack. But as the number of samples from each class ranges between 2 and 1 million, these selected weights were too high for underrepresented classes, and it did not allow the network to stabilize. The loss value jumped between each epoch and eventually the network settled to output a single value. Therefore, weights are applied according to attack categories.

In the undersampling method, 2000 attacks from each category are selected; however, less than 2000 samples exist for R2L and U2R attacks, therefore all existing samples from these categories are included. In addition to anomalous samples, normal samples are added to the undersampled dataset so that the total number of anomalous samples and normal samples are equal to each other.

Table 5.5. *Binary Classification Results*

Model	*Accuracy*	*Precision*	*Recall*	*F1-score*
Vanilla FCN	0.931	0.918	0.996	0.955
FCN w/ Weights	0.927	0.913	0.996	0.9532
FCN w/ Sampling	0.935	0.927	0.9923	0.958
LSTM	0.941	0.929	0.998	0.962
LSTM w/ Sampling	0.942	0.939	0.989	0.963
TCN	0.942	0.929	0.998	0.962
TCN w/ Sampling	0.941	0.939	0.988	0.963

The binary metrics get worse when using the sampling method. In addition to that, sampling-based methods have higher false alarm rates. However, without sampling the networks cannot detect the R2L and U2R attack categories and can only detect DoS and probe attacks. Overall, it can be concluded that sampling models are better than non-sampling models.

Table 5.6. *Binary Classification Detection Rates*

Model	*Normal*	*DoS*	*Probe*	*R2L*	*U2R*	*Avg.*
Vanilla FCN	0.986	0.9614	0.93	0.308	0.285	0.695
FCN w/ Weights	0.987	0.948	0.879	0.4246	0.785	0.805
FCN w/ Sampling	0.970	0.940	0.958	0.737	0.828	0.887
LSTM	0.993	0.992	0.843	0.062	0.185	0.615
LSTM w/Sampling	0.958	0.970	0.892	0.517	0.7	0.807
TCN	0.993	0.992	0.839	0.077	0.328	0.646
TCN w/ Sampling	0.953	0.961	0.874	0.518	0.714	0.806

Lastly, for zero-day attack comparison, the detection rates of attacks that only exist in the test set are compared in Table 5.7. Although there are 237594 DoS samples in the testing set, only 6555 of them are unique to the testing set. Out of these 6555 samples 5000 of them belong to the same attack type: mailbomb. Since the neural network cannot detect this attack, their DoS detection rates are very low. However, they can detect more than 99% of the DoS attacks that exist in the training dataset, thus very high DoS detection rates are achieved. LSTM and TCN models achieve two times better performance in detecting DoS attacks, probably because of their abilities to extract sequential patterns. However, for all other attack types the FCN model is superior to the others and has the best average detection rate. Overall, the detection rates of unknown attacks are lower than the detection rate of known attacks except LSTM networks' probe attack detection rate.

Table 5.7. *Zero-day Attack Detection Rates*

Model	*DoS*	*PROBE*	*R2L*	*U2R*	*Avg*
FCN w/ Sampling	0.117	0.888	0.616	0.613	0.558
LSTM w/Sampling	0.242	0.896	0.449	0.354	0.486
TCN w/ Sampling	0.223	0.856	0.471	0.387	0.484

5.4. Autoencoder Models

Similar to classification models, optimal hyper-parameters for autoencoders were also found by trial and error after the training dataset was split into a training set and two validation sets. An architecture containing a 3-layer encoder followed by a 3-layer decoder was optimal for all autoencoders which can be seen in Figure 5.4. The first layer after the input layer has 300 neurons, which is larger than the number of features in the processed KDD'99 dataset, which has 122 features. This allows autoencoder to model the features better. The latent layer has 10 neurons, which means autoencoder represents the input with 122 features using 10 features.

For FCN and TCN autoencoders the ReLU activation function is used in all layers except the layer directly after the latent view. However, for LSTM tanh activation function is used as it was originally designed by Graves [33]. Using ReLU with LSTM caused the "Dying ReLU" problem. A large gradient can cause a weight update to a ReLU neuron so that it will never activate again regardless of the input [52].

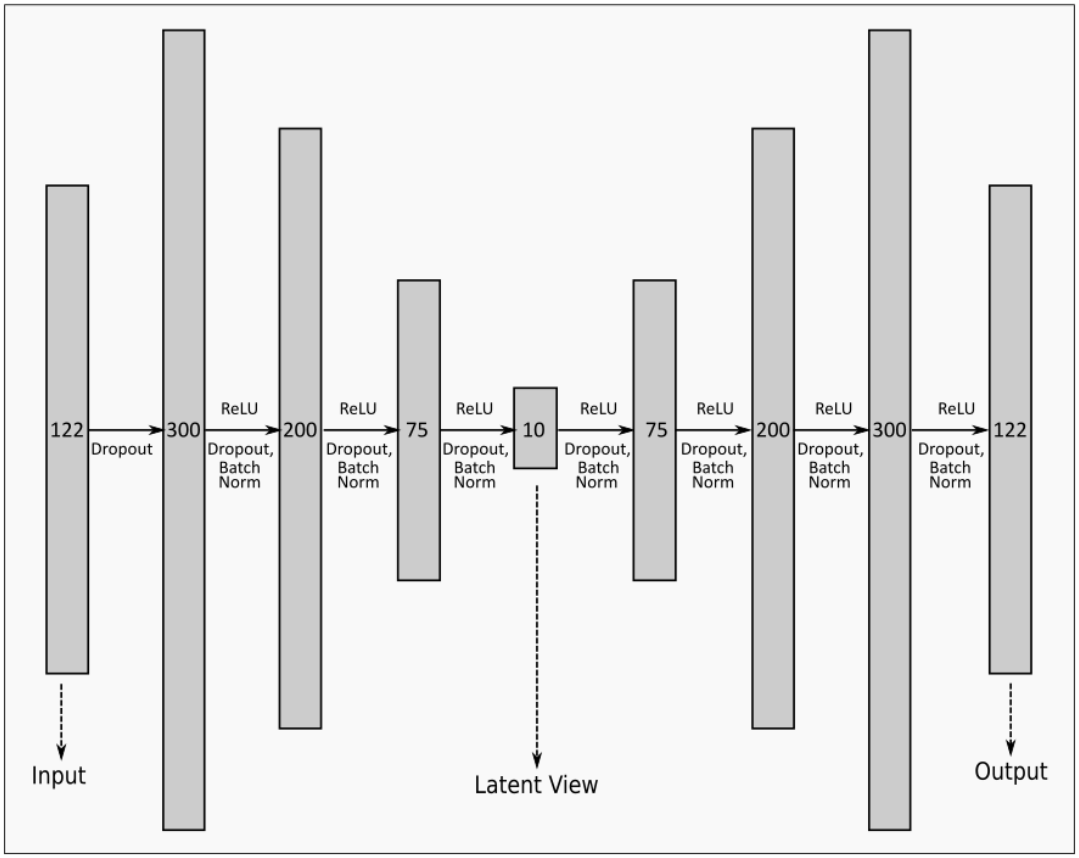

Figure 5.3. Autoencoder Model

For LSTM and TCN autoencoders fixed size windows were selected after experimentations with window sizes of 32, 64, 128 and 256. For both LSTM and TCN, the sequence with 64 samples has the optimal sequence length.

Since LSTM-AE and TCN-AE models make predictions per sequence instead of samples, the calculation of metrics is different for these models. The number of sequences in the test set is equal to the difference between the number of elements in the test set and the selected window size. The binary metrics are calculated using the actual and predicted label of the sequences.

Table 5.8. *Binary Metrics for Autoencoders*

Model	*Accuracy*	*Precision*	*Recall*	*F1-score*
FCN-AE	0.953	0.982	0.959	0.970
LSTM-AE	0.916	0.940	0.971	0.955
TCN-AE	0.930	0.979	0.948	0.963

The detection rates for attacks in the sequences are determined by whether the sequence, which has the attack as its middle element, is identified correctly or not. However, if an attack that is harder to detect is next to an easier to detect attack, it is impossible to say whether the attack can be successfully detected or not. Fortunately, harder to detect U2R and R2L attacks are not placed closer to the other attacks, therefore the detection rate information represents whether the attack can be detected or not accurately.

Table 5.9. *Detection Rates for Autoencoders*

Network	*Normal*	*DOS*	*PROBE*	*U2R*	*R2L*	*Avg.*
FCN-AE	0.962	0.994	0.999	0.376	0.928	0.851
LSTM-AE	0.972	0.940	0.888	0.277	0.228	0.661
TCN-AE	0.936	0.972	0.935	0.310	0.285	0.680

The fully connected autoencoder can detect almost every attack in each category perfectly except for U2R attacks. The reason why it cannot detect U2R attacks is that the features of some of these attacks are very similar to some of the normal samples. In particular, there is an attack called “snmpgetattack” in the test set, for which normal samples with the exact same features exist in the training set. Removing these from the normal training data increases the detection accuracy for U2R attacks. Anomaly detection systems rely on the assumption that anomalies have distinctive properties and this assumption does not hold for U2R feature values.

LSTM-AE and TCN-AE have lower accuracy and f-1 scores in comparison to FCN-AE and do not learn to detect rare attacks. The possible reason for this is that the dataset might not be fit for sequence-based anomaly detection. Most of the previous works regarding LSTM-AE based anomaly detection systems were done on datasets, whose inputs consist of sequences collected from a single source in a continuous manner such as sensor data, stock market rates, seasonal KPI data and so on. Moreover, these values change in a continuum [48], [49]. In the KDD'99 dataset, the samples are listed in sequential order. However, two samples that are next to each other in the dataset might not be related to each other. In a computer network, multiple packets are transmitted between multiple computers at the same time. These packets might have no relation to each other besides occurring simultaneously. Thus, processing them in the same sequence would only create noise. In addition to that, the samples in the dataset have no identifying information such as IP addresses or port numbers. Because of this, the samples that are related to each other cannot be detected. In a way, the KDD'99 dataset is not a long sequence, but a list of shorter sequences sorted together.

In addition to that, the amount of data might not be enough for these models. These architectures are more complex than fully connected architectures, which in general require more data. Adding this to the complexity of the KDD'99 dataset sequences, the architectures might need more data to capture the characteristics of the dataset.

The second possible reason is that sequence-based models might not be necessary. The KDD'99 dataset has features, which contain information about the packets that arrived in the last 2 second window and the last 100 packets, which help detection of sequential attacks.

In general, the best performing model is FCN-AE, since it has the best accuracy, f1-score and average attack detection rates. The only criterion for which this architecture is not the best is the U2R attack detection rate and false alarm rate. The LSTM classifier has the lowest false alarm rate.

5.5. Performance Comparison with SVM

SVM is one of the most successful machine learning algorithms in intrusion detection. SVM can be used for supervised training while a modified version of the algorithm, One-Class SVM (OC-SVM) can be used as an unsupervised training method. [16] compared the performance of SVM and ANN on the KDD'99 dataset in 2002. The results showed that the detection results of SVM are better than those of deep learning. Therefore, SVM and OC-SVM will be used to measure the success of the deep learning algorithms on metrics such as accuracy, f1-score, detection rate and run time.

SVM is a supervised classifier, which takes labelled data in which labels are categorical. SVM determines an optimal hyper-plane, which separates the training data according to their classes. There are many hyper-planes, which separate the data, but the optimal hyper-plane is described as the hyper-plane that maximizes the margin between the classes. The hyperplane separates the data linearly, which makes SVM not applicable for tasks more complex than linear problems. However, by using kernel functions the mapping for more complex problems can be done [53].

One-class SVM (OC-SVM) is a modified version of SVM, which is trained using only data that belongs to one class. OC-SVM separates the input data from the origin in the feature space and maximizes the distance of the hyperplane separating the data and origin. As a result, the algorithm captures the regions in the input space, where the input data lives. Given new data, the algorithm returns whether the data belongs to the region of the input class or not [54].

There are two main advantages SVM has over the deep learning-based intrusion detection algorithms. Firstly, the loss function of SVM algorithm, hinge loss, is a concave function, which means it has a single minimum value that the training algorithm eventually is able to find. Deep learning algorithms use non-convex loss functions, which have local minima that training algorithms can get stuck at [55]. Secondly, the SVM classifier is less affected from dataset imbalance than deep

learning based classifiers, since SVM only considers samples that are closer to boundaries that separates the different classes, which are called support vectors.

SVM and OC-SVM algorithms were used on KDD'99 to compare the results with the deep learning models. SVM classification performances are shown in Table 5.10. The results show that the performances of deep learning classifiers are comparable to deep learning classifiers when it comes to accuracy and f1-scores, which are in the range of 0.935-0.941 and 0.958-0.964 respectively. However, FCN-AE outperforms both SVM and OC-SVM with 0.953 accuracy and 0.97 f1-score.

Table 5.10. *SVM Binary Classification Results*

Model	*Accuracy*	*Precision*	*Recall*	*F1-Score*
SVM	0.939	0.938	0.990	0.963
OC-SVM	0.941	0.941	0.988	0.964

The detection rates of SVM and OC-SVM for different attack categories are given in Table 5.11. Unlike deep learning-based classifiers, SVM can detect infrequent attacks without additional measures such as undersampling. Thus, SVM might be able to perform better for practical intrusion detection than deep learning-based classifiers.

OC-SVM has the highest detection rate for U2R type attacks with 0.832 between all tested algorithms. However, FCN has a higher average detection rate with 0.887 in comparison to OC-SVM's 0.880. In addition to that, besides U2R attacks, for all attack types FCN-AE has higher detection rates than OC-SVM while having a lower false positive rate. In conclusion, FCN-AE outperforms SVM and OC-SVM algorithms in every metric other than detection of U2R attacks.

Table 5.11. *SVM Attack Detection Rates*

Model	*Normal*	*DoS*	*PROBE*	*U2R*	*R2L*	*Avg*
SVM	0.980	0.960	0.822	0.438	0.667	0.734
OC-SVM	0.934	0.980	0.990	0.832	0.667	0.880

5.6. Performance Comparison with Existing Works

In this section the performance of the deep learning-based algorithms is compared with existing works from the literature. All selected papers use the KDD'99 training set for training and the complete KDD'99 dataset for testing. For comparison of machine learning methods, the work of [56] is selected, where performances of different algorithms are compared. For deep learning methods, the results are collected from various papers where the KDD'99 dataset is used.

Table 5.12. *Accuracy and f1-Scores of Existing Works*

Algorithm	*Accuracy*	*f1-score*	*Other Scores*
AdaBoostM1	0.915	0.945	-
BayesNet	0.916	0.945	-
Decision Table	**0.947**	**0.966**	-
J48	0.934	0.958	-
MLP	0.918	0.947	-
Naïve Bayes	0.914	0.944	-
OneR	0.907	0.939	-
Random Forest	0.924	0.950	-
RBF Network	0.852	0.900	-
SGD	0.922	0.945	-
LSTM with RTRL [21]	0.938	0.96	-
DBN on stacked RBM [24]	-	-	DR= 0.935 Recall = 0.923
AE with softmax classifier [25]	**0.956**	**0.969**	FPR= 0.42%
Semi-supervised DRBM [57]	0.940	-	-
AE for dimensionality reduction and RBM [58]	0.921	-	Recall= 0.922 FPR= 1.58%
FCN (This work)	0.935	0.958	-
FCN-AE (This work)	**0.953**	**0.970**	-

In comparison to existing methods, only one work has higher accuracy rates than FCN-AE, which has 0.956% accuracy in comparison to 0.953% accuracy rate of FCN-

AE. However, FCN-AE has a higher f1-score with 0.97 in comparison to 0.969 [25]. The work also uses an autoencoder for intrusion detection. However, unlike the model in this thesis, after training the autoencoder [25], it adds a softmax layer to the end of the network and trains the network in a supervised manner. Overall, the performances of both algorithms are similar to each other. Overall, autoencoders seem to be the algorithm with the highest accuracy and f1-scores among the deep learning algorithms and hybrid deep learning algorithms, which combine deep learning algorithms with other algorithms.

In general, papers which focuse on anomaly-based intrusion detection algorithms do not report individual detection rates for different attack categories. However, there are several works that use classification methods, which uses deep learning algorithms to make 5-class classifications in which each attack category and normal samples are used as classes. Table 5.13 reports detection rates from existing algorithms, where 2-class anomaly detection or 5-class detection algorithms are used on the KDD'99 dataset.

Table 5.13. shows that FCN-AE has the highest detection rates for DoS, Probe and R2L attacks and FCN has the highest average attack detection rate. However, OC-SVM has the highest detection rate for U2R attacks. In general, the results show that majority of the existing algorithms are not capable of detecting U2R and R2L attacks. Among the previous works, the highest U2R detection rate is 0.44, while the highest detection rate for this attack type is 0.832, which is obtained using OC-SVM. Among the previous works the highest R2L detection rate is 0.182 while the highest R2L detection rate is 0.928, obtained with the FCN-AE algorithm.

Table 5.13. *Detection Rates from Existing Works*

Model	*Normal*	*DOS*	*PROBE*	*U2R*	*R2L*	*Avg*
C. 4.5 [59]	0.995	0.971	0.833	0.132	0.084	0.603
Pnrule [60]	0.995	0.969	0.732	0.066	0.107	0.574
C. 4.5 w/ Oversampling [61]	0.995	0.941	0.747	0.257	0.062	0.600
Random Forest w/ Oversampling [61]	0.995	0.942	0.746	0.243	0.106	0.606
MLP [62]	0.984	0.970	0.860	0.143	0.119	0.615
Boosted J48 [62]	0.995	0.969	0.920	0.118	0.171	0.635
General Regression NN [62]	0.911	0.993	0.853	0.440	0.128	0.665
Boosted Modified Probabilistic NN [62]	**0.998**	0.984	0.917	0.227	0.182	0.662
ANN [61], [63]	0.993	0.939	0.741	0.129	0.129	0.586
LSTM [21]	0.995	**0.994**	0.784	0.031	0.171	0.595
SVM *(This work)*	0.980	0.960	0.822	0.438	0.667	0.734
OC-SVM *(This work)*	0.934	0.980	0.990	**0.832**	0.667	0.880
Vanilla FCN *(This work)*	0.986	0.9614	0.93	0.308	0.285	0.695
FCN w/ Sampling *(This work)*	0.970	0.940	0.958	0.737	0.828	**0.887**
FCN-AE *(This work)*	0.962	**0.994**	**0.999**	0.376	**0.928**	0.851

CHAPTER 6

TRANSFER LEARNING

6.1. Introduction

This chapter summarizes the results of transfer learning experiments on neural networks, which are trained on the KDD'99 dataset. The trained networks were then used to detect intrusions on the IDS 2017 dataset, which was created by the Canadian Institute for Cybersecurity in 2017. This dataset is almost twenty years newer than the KDD'99 dataset, therefore, there are some attacks that did not exist in the dataset as well as non-anomalous data, which is extracted from network services that did not exist before.

6.2. CI IDS 2017 Dataset

The dataset is a generated dataset, which used a profiling system to create abstract behaviors of 25 users based on several network services such as HTTP, FTP, email protocols, etc. [64].

The created dataset spans 5 work days from Monday to Friday during work hours. In the Monday dataset, there exist only non-anomalous data while other days contain anomalous data too. There are 17 different attacks, which include zero-day attacks and multi-step attacks in the dataset. The attacks belong to the categories below:

Brute Force Attack: This is an attack, where the attacker tries to send as many messages as possible for purposes such as cracking a password or finding hidden content. If this attack was in the KDD'99 dataset, it would be labelled as a PROBE attack.

Botnet: A Botnet is a number of devices each of which runs one or more bots. The bots can be used for several purposes including participating in DoS attacks, phishing, email scams, Bitcoin mining, etc. [65].

DoS Attack: DoS attacks aim to overload the network by sending too many packets. Similar to the KDD'99 dataset, DoS attacks are the attacks with the largest number of samples in the network.

Heartbleed Attack: This attack allows exploitation of a bug in the OpenSSL library, which provides encryption for Internet applications such as Web, e-mail and VPN. This bug allows anyone on the Internet to get unauthorized access to data on systems that use problematic versions of the OpenSSL library [66].

Web Attack: Web attacks are attacks, which exploit vulnerabilities in Web applications. Web attacks in this dataset include Cross-Site Scripting, which allows the attacker to inject scripts into websites and SQL injection, which allows the attacker to run SQL queries on the victim's database.

Infiltration Attack: Infiltration is a multi-step attack, where a vulnerable software is exploited to gain access and create a backdoor on the victim. After the backdoor is created different attacks can be conducted such as port scan, Nmap and so on.

The dataset is available to the public as both raw files and labelled tabular data. The labelled data has different features than the KDD'99 dataset. Therefore, feature extraction is required to convert the data into the same format as the KDD'99 dataset.

The Zeek Security Monitor tool was used to extract features from individual packets, then a script was used to convert the output of the tool to create a tabular dataset with KDD'99 dataset features [67].

For features that cannot be extracted, such as "num_failed_logins" and "logged_in", the values are filled in with 0s.

Originally the created dataset consisted of 3612571 samples, where 3271619 are normal data and 340952 are attacks. However, about 70% of the extracted data were

DNS queries. These samples are dropped from the dataset, and as a result the final number of data points in the dataset is 1055884.

6.3. CI IDS 2017 Test Results

Experiment results from previous chapters showed that fully connected classifier and fully connected autoencoder networks perfrom better than LSTM and TCN models. Because of this, fully connected architectures used in this section to be trained using using CI IDS 2017 data.

The classifier is trained by using 5-fold cross-validation. While dividing the dataset into folds, the ratios of each attack is kept the same as the original dataset. The autoencoder model is trained using only normal samples from the same training datasets used for the classifier above. The validation datasets are split into two similar datasets to decide the anomaly threshold. The results are given below:

Table 6.1. *CI IDS Dataset Results*

Model	*Accuracy*	*Precision*	*Recall*	*F1-score*
FCN	0.997	0.985	0.985	0.984
FCN-AE	0.987	0.950	0.915	0.932

Table 6.2. *Detection Rates*

Attack Type	*Number of Samples*	*Classifier DR*	*Autoencoder DR*
Brute Force	6972	0.791	0.807
DOS	330398	0.993	0.960
Heartbleed	8	0.000	0.333
Web Attack	2066	0.388	0.242
Infiltration	59	0.111	0.547
Botnet	1449	0.374	0.645
Normal	3271619	0.999	0.990

In CI IDS 2017 datasets case, unlike KDD'99, the classifier performed better than the autoencoders. The possible reason for this is that this dataset does not suffer from imbalance problems as much as the KDD'99 dataset does. In balanced datasets it is expected that supervised methods will perform better than the unsupervised methods. However, similarly to the KDD'99 dataset autoencoder outperformed the classifier in detection of rare attacks.

6.4. Transfer Learning

There are plenty of works in the field of deep learning-based intrusion detection. However, in all of these works the datasets used for training and testing are collected from the same environment. The purpose of this experiment is to test whether a neural network trained using a dataset created in an environment can be used to detect anomalies in another environment. In the literature there exists one work, which applies transfer learning to the intrusion detection field. The work trains an SVM on the KDD'99 dataset and uses the model to detect anomalies in the Kyoto 2006 intrusion detection dataset. The paper removes infrequent attacks from both the training and test datasets and reports 99% detection accuracy in the target dataset [68], [69].

This section includes experiments including testing the network models, which are trained using KDD'99 and CI IDS 2017 with the other dataset. The datasets are almost 20 years apart and the average normal model has changed in the past 20 years.

Initial experiments showed that for both classifiers and autoencoders, these approaches do not work as all models classified every item in the other dataset as an anomaly. However, assuming the autoencoder does reconstruct the normal items from the other dataset in a similar way, the reconstruction error would be similar for all normal items.

The table shows the detection rates of the KDD'99 dataset according to the autoencoder trained on the CI IDS dataset. The threshold used is the mean of the reconstruction errors of the elements of the dataset.

Table 6.3. *FCN-AE Detection Rates*

Attack	*Detection Rate*
Normal	0.999
DoS	0.825
Probe	0.509
U2R	0.0001
R2L	0.071

For this experiment, the accuracy is 0.784, precision 0.732, recall is 0.999 and f1-score is 0.845.

The table below shows the detection rates of the CI IDS dataset according to autoencoder trained on the KDD'99 dataset.

Table 6.4. *FCN-AE Detection Rates*

Attack	*Detection Rate*
Brute Force	0.002
DoS	0.630
Heartbleed	0.000
Web Attack	0.000
Infiltration	0.000
Botnet	0.000
Normal	0.754

Accuracy is 0.705, precision is 0.598, recall is 0.529 and f1-score is 0.61. The results show that the autoencoders trained on the other dataset can only detect DoS attacks. For more complex attacks this approach does not work.

The difference between the results of the networks is that the first network has almost no false alarms while the second network considers some of the normal data as abnormal. The inspection of false alarms showed that in KDD'99 the false alarms are mainly caused by the messages sent from UDP ports 137 and 138. These ports are

assigned for NetBIOS, which is a protocol used for File and Print Sharing under all current versions of Windows. The way this protocol was implemented allows binding to any connection rather than only the local network. Thus, it can be and was used for malicious activities including the following trojans/backdoors, which also use these ports: Chode, God Message worm, Msinit, Netlog, Network, Qaz. In total there are 101909 samples using this protocol and these samples are the majority of the network errors.

The results of the transfer learning experiments show that direct transfer of the intrusion detection solution is not possible across network systems. However, both of the used datasets are simulations, which were created twenty years apart.

CHAPTER 7

CONCLUSION

7.1. Summary

In this thesis, supervised and unsupervised anomaly detection methods were tested on a well-known intrusion detection dataset. The results showed that unsupervised methods are better at detecting infrequent attacks while supervised methods perform better when there are enough samples of the attack in the dataset. Intrusion detection is a field where the anomalies are constantly evolving in both quantity and complexity, it is impossible to collect data about all existing and future malicious activities. Therefore, unsupervised methods are better suited for the intrusion detection task.

The anomaly-based methods work on the assumption that the anomalous data is different from the normal data. These types of attacks might not have differentiable features in comparison to normal data, and then anomaly detection cannot be used. In the KDD'99 dataset, the fully connected autoencoder is able to detect all anomalies except the U2R samples, which have the same features as the normal samples. In this case the problem is not the algorithm, but the features extracted from the dataset. For traditional machine learning algorithms, the way the data are represented affects the performance of the network heavily, since these algorithms cannot extract complex patterns while deep learning algorithms can extract latent features from the representation because of the deep layered model. The algorithms are not tested directly on the network packets, but the features extracted from them. Therefore, the success of the algorithms might be limited by the feature engineering process.

The results show that the simpler fully connected models work better than sequential models. The possible reason for that might be that sequential models are not a good

fit for this task, or the representation of the data might prevent these models from capturing temporal dependencies.

The transfer learning section of this thesis shows that the model trained on a network cannot be used on another network to detect infrequent attacks since the baseline normal activities are different from each other. The datasets are 18 years apart from each other. In the newer dataset the packets are larger, and durations of connections are longer. In addition to that, in the IDS 2017 data, there are packets from different services. However, the networks were able to capture some of the attacks, which can mean that these algorithms can be improved. Also, previously mentioned limitations of the feature selection process can also limit the success of the transfer learning process.

The experiments in this thesis showed the unique challenges of intrusion detection tasks. Firstly, anomalies change over time, which itself is not unique to the intrusion detection field. For example, the anomalous behavior changed in Maroochy water treatment plant and because of this it could not be detected [70]. However, in the intrusion detection field anomalies are actively created by skilled malicious actors in sophisticated ways. The second challenge of anomaly-based intrusion detection is that not only the anomalous data changes but normal data changes too. The comparison of KDD'99 dataset and CI IDS 2017 dataset showed that there are many services available in the 2017 dataset. In addition to that, the average numbers of source and destination bytes in each connection increased in 2017.

Secondly, the volume of data collected from a computer network is in general larger than those for many anomaly detection tasks. For intrusion detection each connection arriving at an IDS must be processed. The very common 1-Gb/s Ethernet interface can transmit up to 125,000,000 bytes/s, which means between 80,000 to 1,500,000 packages per second can be transmitted using the interface [71]. The raw data collected for CI IDS Dataset is about 10 GB per day. Processing computer network data is resource-intensive in comparison to processing a dataset collected less

frequently e.g. hourly, weekly, monthly. In addition to that, data collected from a computer network is collected from multiple devices in the network, most of which have no relation to each other.

7.2. Future Works

In this thesis, the performances of deep learning algorithms for intrusion detection models are evaluated using the KDD'99 dataset. KDD'99 consists of 41-column tabular data extracted from raw network packets. The experiments show that using selected features, the algorithms were able to detect DoS attacks with high detection rates. However, for U2R attacks the features were similar to normal samples and because of this the algorithms were not able to differentiate between them. Even in transfer learning experiments, the algorithms were only able to detect DoS attacks in the test set. The reason for this is that 18 out of 41 features are information extracted from the packets collected in the last 2 seconds and the last 100 packets before the dataset. Since DoS attacks are sent in a higher speed than normal and other attack samples, these 18 features distinguish between DoS attacks and other samples. As a result, the algorithms can detect these attacks easier than other attack categories. For machine learning algorithms, the performance of the algorithms is dependent on the representation of the data since these algorithms cannot extract simpler representations from complex representations. However, deep learning algorithms can learn from complex representations. Therefore, instead of tabular data more complex representations can be used to extract properties of the normal and attack samples. Even raw network packets can be used as an input to NLP algorithms and intrusion detection can be done without feature engineering.

The experiments in this thesis showed that LSTM and TCN do not perform as well as simple fully connected architectures. The reason for this is that although the used datasets are ordered in a sequential manner, the samples next to each other are not always related to each other. This is because the samples are collected from multiple connections across the network between different source and destinations using

different services. In addition to that, since the identifying information such as IP addresses and port numbers of the source and destination are excluded from the dataset, it is impossible for the algorithms to identify packets that belongs to the same connection. However, by changing the representation of the data to include this information, sequential dependencies between the samples can be used to detect intrusions.

Secondly, collection of anomalous samples for non-sequential attacks is an important task for development of intrusion detection algorithms. Intrusion detection datasets are naturally imbalanced since collected data contains mostly normal samples. However, the majority of the anomalous data belong to sequential attacks such as DoS. The number of samples of other types of attacks are low. For example, in the KDD'99 dataset such attacks make up less than 0.5% of the dataset. In the CI IDS 2017 dataset, there are 8 samples of Heartbleed attacks and 59 samples of Infiltration in comparison to 330398 samples of DoS attacks. Collection of the data from such attacks would be useful in intrusion detection.

Lastly, transfer learning experiments were done on two datasets collected in 1999 and 2017 respectively which makes the datasets 18 years apart from each other. In these years, several new technologies and services were introduced, which did not exist when the KDD'99 dataset was created. The algorithms were trained using the KDD'99 dataset, resulting in the classification of some of the new services as anomalies. If datasets which are created in different environments at the same time are used for transfer learning, the algorithms might perform better.

REFERENCES

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, 2019.

[2] The Internet Society's Online Trust Alliance (OTA), "2018 Cyber Incident & Breach Trends Report," 2019.

[3] "The Purpose of Intrusion Detection & Prevention Systems," 2017. [Online]. Available: https://www.7sec.com/blog/the-purpose-of-intrusion-detection-and-prevention-systems/].

[4] I. Security and T. Report, "ISTR Symantec," no. April, p. 10, 2017.

[5] I. Rijnetu, "Security Alert: MS Office Zero Day and DNS Vulnerabilities Can Impact Users," 2017. [Online]. Available: https://heimdalsecurity.com/blog/security-alert-microsoft-vulnerabilities-in-office-and-dns/.

[6] L. Bilge and T. Dumitras, "Before we knew it," no. October 2012, p. 833, 2012.

[7] D. E. Denning, "Intrusion Detection Model," *IEEE Trans. Softw. Eng.*, 1987.

[8] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," in *2016 International Conference on Platform Technology and Service, PlatCon 2016 - Proceedings*, 2016.

[9] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Trans. Emerg. Top. Comput. Intell.*, 2018.

[10] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction," in *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017*, 2017.

[11] J. Kim, N. Shin, S. Y. Jo, and S. H. Kim, "Method of intrusion detection using deep neural network," in *2017 IEEE International Conference on Big Data and Smart Computing, BigComp 2017*, 2017.

[12] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Syst. Appl.*, 2014.

[13] A. Özgür and H. Erdem, "Saldırı tespit sistemlerinde genetik algoritma kullanarak nitelik seçimi ve çoklu sınıflandırıcı füzyonu," *J. Fac. Eng. Archit. Gazi Univ.*, 2018.

[14] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*. 2015.

[15] J. Shun and H. A. Malki, “Network Intrusion Detection System Using Neural Networks,” 2009.

[16] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using neural networks and support vector machines,” 2003.

[17] Jwg-sheng Xue, Ji-zhou Sun, and Xu Zhang, “Recurrent network in network intrusion detection system,” 2005.

[18] J. Skaruz and F. Seredynski, “Recurrent neural networks towards detection of SQL attacks,” in *Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM*, 2007.

[19] S. S. Roy, A. Mallik, R. Gulati, M. S. Obaidat, and P. V. Krishna, “A deep learning based artificial neural network approach for intrusion detection,” in *Communications in Computer and Information Science*, 2017.

[20] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for Network Intrusion Detection in Software Defined Networking,” in *Proceedings - 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM 2016: Green Communications and Networking*, 2016.

[21] R. C. Staudemeyer, “Applying long short-term memory recurrent neural networks to intrusion detection,” no. 56, pp. 136–154, 2015.

[22] C. Yin, Y. Zhu, J. Fei, and X. He, “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks,” *IEEE Access*, 2017.

[23] A. Elsherif, “Automatic Intrusion Detection System Using Deep Recurrent Neural Network Paradigm,” *J. Inf. Secur. Cybercrimes Res.*, 2018.

[24] N. Gao, L. Gao, Q. Gao, and H. Wang, “An Intrusion Detection Model Based on Deep Belief Networks,” in *Proceedings - 2014 2nd International Conference on Advanced Cloud and Big Data, CBD 2014*, 2015.

[25] F. Farahnakian and J. Heikkonen, “A deep auto-encoder based approach for intrusion detection system,” *Int. Conf. Adv. Commun. Technol. ICACT*, vol. 2018-Febru, pp. 178–183, 2018.

[26] A. H. Mirza and S. Cosan, “Computer network intrusion detection using sequential LSTM Neural Networks autoencoders,” in *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018*, 2018.

[27] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *Comput. Secur.*, 2012.

[28] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, 1958.

[29] S. Ruder and B. Plank, “Strong Baselines for Neural Semi-Supervised Learning under Domain Shift,” 2019.

[30] R. J. Williams and D. Zipser, “Gradient-based learning algorithms for recurrent networks and their computational complexity,” *Back-propagation Theory, Archit. Appl.*, 1995.

[31] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, “Malware classification with recurrent networks,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2015.

[32] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, 1997.

[33] A. Graves, *Supervised Sequence Labeling with Recurrent Neural Networks*. 2013.

[34] G. Hoffman, “Introduction to LSTMs with TensorFlow,” 2018. [Online]. Available: https://www.oreilly.com/ideas/introduction-to-lstms-with-tensorflow.

[35] S. Bai, J. Z. Kolter, and V. Koltun, “Convolutional Sequence Modeling Revisited,” in *ICLR Workshop*, 2018.

[36] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks: A unified approach to action segmentation,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[37] S. Turukin, “Wavenet,” 2017. [Online]. Available: http://sergeiturukin.com/2017/03/02/wavenet.html.

[38] S. Hettich and S. D. Bay, “The UCI KDD Archive,” *University of California, Department of Information and Computer Science*, 1999. .

[39] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009.

[40] M. V. Mahoney and P. K. Chan, “An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2003.

[41] T. Schaul, I. Antonoglu, and D. Silver, “No Title,” *Pro-ceedings Int. Conf. Learn. Represent.*, 2014.

[42] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning arXiv : 1705 . 08292v2 [stat . ML] 22 May 2018,” in *31st Conference on Neural Information Processing Systems (NIPS,* 2017.

[43] J. Heaton, “Programming Neural Networks in Java,” *Java Dev. J.,* 2002.

[44] T. Masters, *Practical Neural Networks Recipes in C++*. 1993.

[45] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, “Long Short Term Memory Networks for Anomaly Detection in Time Series,” *Eur. Symp. Artif. Neural Networks,* 2015.

[46] I. Sutskever and Ilya Sutskever, “Training Recurrent Neural Networks,” *Ph.D thesis,* 2013.

[47] R. J. Williams and J. Peng, “An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories,” *Neural Comput.,* 2008.

[48] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, “LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection,” Jul. 2016.

[49] H. Xu *et al.*, “Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications,” 2018.

[50] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Comput.,* 2002.

[51] I. Goodfellow, B. Yoshua, and A. Courville, “Deep Learning.” [Online]. Available: http://www.deeplearningbook.org/. [Accessed: 06-Aug-2019].

[52] K. Parthy, “CS231n Convolutional Neural Networks for Visual Recognition,” *Stanford Univ. Course cs231n,* 2018.

[53] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Mach. Learn.,* 1995.

[54] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Piatt, “Support vector method for novelty detection,” in *Advances in Neural Information Processing Systems*, 2000.

[55] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, “Are Loss Functions All the Same?,” *Neural Comput.,* 2004.

[56] H. Erdem, “A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015,” *PeerJ Prepr.,* 2016.

[57] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, “Network anomaly detection with the restricted Boltzmann machine,” *Neurocomputing*, 2013.

[58] Y. Li, R. Ma, and R. Jiao, “A hybrid malicious code detection method based on

deep learning," *Int. J. Secur. its Appl.*, 2015.

[59] B. Pfahringer, "Winning the KDD99 Classification Cup: Bagged Boosting," *ACM SIGKDD Explor.*, 2000.

[60] R. Agarwal and M. V. Joshi, "PNrule: A New Framework for Learning Classifier Models in Data Mining (A Case-Study in Network Intrusion Detection)," 2001.

[61] T. Chandak, S. Shukla, and R. Wadhvani, "An analysis of 'A feature reduced intrusion detection system using ANN classifier' by Akashdeep et al. expert systems with applications (2017)," *Expert Syst. Appl.*, 2019.

[62] T. P. Tran, T. T. S. Nguyen, P. Tsai, and X. Kong, "BSPNN: Boosted subspace probabilistic neural network for email security," *Artif. Intell. Rev.*, 2011.

[63] Akashdeep, I. Manzoor, and N. Kumar, "A feature reduced intrusion detection system using ANN classifier," *Expert Syst. Appl.*, 2017.

[64] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018.

[65] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. Van Steen, and N. Pohlmann, "On botnets that use DNS for command and control," in *Proceedings - 2011 7th European Conference on Computer Network Defense, EC2ND 2011*, 2012.

[66] Z. Durumeric *et al.*, "The matter of heartbleed," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2014.

[67] "The Zeek Network Security Monitor." [Online]. Available: https://www.zeek.org/.

[68] Z. Taghiyarrenani, A. Fanian, E. Mahdavi, A. Mirzaei, and H. Farsi, "Transfer learning based intrusion detection," in *2018 8th International Conference on Computer and Knowledge Engineering, ICCKE 2018*, 2018.

[69] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation," in *Proceedings of the 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2011*, 2011.

[70] D. Ramotsoela, A. Abu-Mahfouz, and G. Hancke, "A survey of anomaly detection in industrial wireless sensor networks with critical water system infrastructure as a case study," *Sensors (Switzerland)*, 2018.

[71] C. S. R. & Operations., "Bandwidth, Packets Per Second, and Other Network Performance Metrics." [Online]. Available:

https://www.cisco.com/c/en/us/about/security-center/network-performance-metrics.html.

APPENDICES

A. KDD'99 Dataset Features

duration: continuous.

protocol_type: symbolic.

service: symbolic.

flag: symbolic.

src_bytes: continuous.

dst_bytes: continuous.

land: symbolic.

wrong_fragment: continuous.

urgent: continuous.

hot: continuous.

num_failed_logins: continuous.

logged_in: symbolic.

num_compromised: continuous.

root_shell: continuous.

su_attempted: continuous.

num_root: continuous.

num_file_creations: continuous.

num_shells: continuous.

num_access_files: continuous.

num_outbound_cmds: continuous.

is_host_login: symbolic.

is_guest_login: symbolic.

count: continuous.

srv_count: continuous.

serror_rate: continuous.

srv_serror_rate: continuous.

rerror_rate: continuous.

srv_rerror_rate: continuous.

same_srv_rate: continuous.

diff_srv_rate: continuous.

srv_diff_host_rate: continuous.

dst_host_count: continuous.

dst_host_srv_count: continuous.

dst_host_same_srv_rate: continuous.

dst_host_diff_srv_rate: continuous.

dst_host_same_src_port_rate: continuous.

dst_host_srv_diff_host_rate: continuous.

dst_host_serror_rate: continuous.

dst_host_srv_serror_rate: continuous.

dst_host_rerror_rate: continuous.

dst_host_srv_rerror_rate: continuous