Otto-Von-Guericke University Magdeburg

Faculty of Electrical Engineering and Information Technology

Institute for Automation Engineering

Chair for Integrated Automation

(to be decided)

Implementation of Intrusion Detection System in CNTK

Supervisor

Dipl.-Ing. Sasanka Potluri

Project by

Akashay Solanki (Matriculation Number - 213739)
Bhavik Vala (Matriculation Number - 213750)
Devanshu Khokhani(Matriculation Number - 214650)
Jay Vala(Matriculation Number - 2153739)

Contents

Abstract **1**

1 Introduction **2**
1.1 Intrusion Detection System 2
1.2 Why should one use intrusion detection system ? 3
1.3 Classification of Intrusion Detection System 3
1.3.1 Host based Intrusion Detection System 3
1.3.2 Network based Intrusion Detection System 3
1.4 Advantages of Host Based Intrusion Detection System 4
1.5 Advantages of Network Based Intrusion Detection System 4
1.6 Limitations of Intrusion Detection Systems 4

2 Deep Neural Networks **6**
2.1 Artificial Neural Network 6
2.2 Artificial Neural Networks: Types 8
2.3 Training a Deep Neural Network 9
2.3.1 Supervised Learning 9
2.3.2 Unsupervised Learning 10
2.3.3 Reinforcement Learning 10
2.3.4 Activation Functions 12
2.3.5 Optimization Algorithm 15
2.4 Challenges in Training a Deep Neural Network 17
2.5 Stacked Autoencoder 17
2.5.1 Structure 17
2.5.2 Types of Autoencoders 18
2.6 Convolutional Neural Networks 19
2.6.1 Convolution 19
2.6.2 Pooling 20
2.7 Recurrent Neural Networks 20
2.7.1 Long Short Term Memory 21
2.8 Greedy Layer Wise Pre-training 23

3 Evaluation and Results **24**
3.1 The Approach 24
3.1.1 Pre-Processing 24
3.1.2 Parameters 26

3.2 Data Set Evaluation 28
3.3 Stacked Autoencoder 29
3.3.1 Results 29
3.3.2 Adam Optimizer 29
3.3.3 RMSProp Optimizer 33
3.3.4 Stochastic Gradient Descent Optimizer 36
3.3.5 Adadelta Optimizer 39
3.4 Convolutional Neural Networks 43
3.4.1 Adam Optimizer 43
3.4.2 RMSProp Optimizer 47
3.4.3 Stochastic Gradient Descent Optimizer 50
3.4.4 Adadelta Optimizer 54
3.5 Recurrent Neural Network(LSTM) 58
3.5.1 Results 58
3.5.2 Adam Optimizer 58
3.5.3 RMSProp Optimizer 62
3.5.4 Stochastic Gradient Descent Optimizer 65
3.5.5 Adadelta Optimizer 69

4 Conclusion **73**

List of Figures

1 Structure of Artificial Neuron 7
2 Structure of Artificial Neural Network 7
3 Feed Forward and Recurrent Neural Network 8
4 Supervised Learning 10
5 Unsupervised Learning(K-Means) 11
6 Reinforcement Learning 11
7 Sigmoid Activation Function 12
8 Step Activation Function 13
9 Rectified Linear Unit Activation Function 13
10 TanH Activation Function 14
11 Softplus Activation Function 15
12 Autoencoder with single hidden layer 18
13 Architecture of Convolutional Neural Network 19
14 Unfolded Recurrent Neural Network 21

15 Accuracy of Adam Optimizer for different activation functions 29
16 Accuracy of Stochastic Gradient Descent Optimizer for different activation functions 36
17 Accuracy of Momentum Optimizer for different activation functions . . 39
18 Accuracy of Adam Optimizer for different activation functions 43
19 Accuracy of RMSProp Optimizer for different activation functions . . . 47
20 Accuracy of Stochastic Gradient Descent Optimizer for different activation functions 50
21 Accuracy of Adadelta Optimizer for different activation functions . . . 54
22 Accuracy of Adam Optimizer for different activation functions 58
23 Accuracy of RMSProp Optimizer for different activation functions . . . 62
24 Accuracy of Stochastic Gradient Descent Optimizer for different activation functions 65
25 Accuracy of Adadelta Optimizer for different activation functions . . . 69

List of Tables

3.1 Distribution of records in Test and Train Set 28
3.2 Configuration of Autoencoder for Greedy Layer-Wise Pre-Training . . . 29
3.3 Confusion matrix for Sigmoid Activation and Adam optimizer for Autoencoder 30
3.4 Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Sigmoid Activation 30
3.5 Confusion matrix for ReLU Activation and Adam optimizer for Autoencoder 30
3.6 Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for ReLU Activation 31
3.7 Confusion matrix for eLU Activation and Adam optimizer for Autoencoder 31
3.8 Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for eLU Activation 31
3.9 Confusion matrix for Softplus Activation and Adam optimizer for Autoencoder 32
3.10 Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Softplus Activation 32
3.11 Confusion matrix for Sigmoid Activation and RMS optimizer for Autoencoder 33
3.12 Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for Sigmoid Activation 33
3.13 Confusion matrix for ReLU Activation and RMSProp optimizer for Autoencoder 33
3.14 Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for ReLU Activation 34
3.15 Confusion matrix for eLU Activation and RMS optimizer for Autoencoder 34
3.16 Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for eLU Activation 34
3.17 Confusion matrix for Softplus Activation and RMS optimizer for Autoencoder 35
3.18 Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for Softplus Activation 35
3.19 Confusion Matrix for Sigmoid Activation and SGD optimizer for Autoencoder 36

3.20 Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for Sigmoid Activation 37
3.21 Confusion matrix for reLU Activation and SGD optimizer for Autoencoder 37
3.22 Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for ReLU Activation 37
3.23 Confusion Matrix for eLU Activation and SGD optimizer for Autoencoder 38
3.24 Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for eLU Activation 38
3.25 Confusion matrix for Softplus Activation and SGD optimizer for Autoencoder 38
3.26 Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for Softplus Activation 39
3.27 Confusion Matrix for Sigmoid Activation and Adadelta optimizer for Autoencoder 40
3.28 Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for Sigmoid Activation 40
3.29 Confusion Matrix for reLU Activation and Adadelta optimizer for Autoencoder 40
3.30 Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for ReLU Activation 41
3.31 Confusion Matrix for eLU Activation and Adadelta optimizer for Autoencoder 41
3.32 Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for eLU Activation 41
3.33 Confusion Matrix for Softplus Activation and Adadelta optimizer for Autoencoder 42
3.34 Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for Softplus Activation 42
3.35 Confusion Matrix for Sigmoid activation and Adam Optimizer for Convolutional Neural Network 43
3.36 Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Sigmoid Activation 44
3.37 Confusion Matrix for ReLU activation and Adam Optimizer for Convolutional Neural Network 44
3.38 Precision, Recall, and F1 Score of Stacked Convolutional Neural Network with Adam Optimizer for ReLU Activation 44
3.39 Confusion Matrix for eLU activation and Adam Optimizer for Convolutional Neural Network 45
3.40 Precision, Recall, and F1 Score of Convolutional Neural Network with Adam Optimizer for eLU Activation 45
3.41 Confusion Matrix for TANH activation and Adam Optimizer for Convolutional Neural Network 45
3.42 Precision, Recall, and F1 Score of Convolutional Neural Network with Adam Optimizer for TanH Activation 46

3.43 Confusion Matrix for Sigmoid activation and RMS Optimizer for Convolutional Neural Network . . . 47
3.44 Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for Sigmoid Activation . . . 48
3.45 Confusion Matrix for ReLU activation and RMS Optimizer for Convolutional Neural Network . . . 48
3.46 Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for ReLU Activation . . . 48
3.47 Confusion Matrix for eLU activation and RMS Optimizer for Convolutional Neural Network . . . 49
3.48 Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for eLU Activation . . . 49
3.49 Confusion Matrix for TanH activation and RMS Optimizer for Convolutional Neural Network . . . 49
3.50 Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for TanH Activation . . . 50
3.51 Confusion Matrix for Sigmoid activation and SGD Optimizer for Convolutional Neural Network . . . 51
3.52 Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for Sigmoid Activation . . . 51
3.53 Confusion Matrix for ReLU activation and SGD Optimizer for Convolutional Neural Network . . . 51
3.54 Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for ReLU Activation . . . 52
3.55 Confusion Matrix for eLU activation and SGD Optimizer for Convolutional Neural Network . . . 52
3.56 Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for eLU Activation . . . 52
3.57 Confusion Matrix for TanH activation and SGD Optimizer for Convolutional Neural Network . . . 53
3.58 Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for TanH Activation . . . 53
3.59 Confusion Matrix for Sigmoid activation and Adadelta Optimizer for Convolutional Neural Network . . . 54
3.60 Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation . . . 55
3.61 Confusion Matrix for ReLU activation and Adadelta Optimizer for Convolutional Neural Network . . . 55
3.62 Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation . . . 55
3.63 Confusion Matrix for eLU activation and Adadelta Optimizer for Convolutional Neural Network . . . 56
3.64 Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation . . . 56
3.65 Confusion Matrix for TanH activation and Adadelta Optimizer for Convolutional Neural Network . . . 56

3.66 Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation 57
3.67 Configuration of Recurrent Neural Network(LSTM) 58
3.68 Confusion matrix for Sigmoid Activation and Adam optimizer for RNN(LSTM) 59
3.69 Precision, Recall, and F1 Score of LSTM with Adam Optimizer for Sigmoid Activation 59
3.70 Confusion matrix for ReLU Activation and Adam optimizer for LSTM . 59
3.71 Precision, Recall, and F1 Score of LSTM with Adam Optimizer for ReLU Activation 60
3.72 Confusion matrix for eLU Activation and Adam optimizer for LSTM . . 60
3.73 Precision, Recall, and F1 Score of LSTM with Adam Optimizer for eLU Activation 60
3.74 Confusion matrix for Softplus Activation and Adam optimizer for LSTM 61
3.75 Precision, Recall, and F1 Score of LSTM with Adam Optimizer for Softplus Activation 61
3.76 Confusion matrix for Sigmoid Activation and RMSProp optimizer for LSTM 62
3.77 Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for Sigmoid Activation 63
3.78 Confusion matrix for ReLU Activation and RMSProp optimizer for LSTM 63
3.79 Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for ReLU Activation 63
3.80 Confusion matrix for eLU Activation and RMS optimizer for LSTM . . 64
3.81 Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for eLU Activation 64
3.82 Confusion matrix for Softplus Activation and RMS optimizer for LSTM 64
3.83 Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for Softplus Activation 65
3.84 Confusion matrix for Sigmoid Activation and SGD optimizer for LSTM 66
3.85 Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for Sigmoid Activation 66
3.86 Confusion matrix for reLU Activation and SGD optimizer for LSTM . . 66
3.87 Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for ReLU Activation 67
3.88 Confusion matrix for eLU Activation and SGD optimizer for LSTM . . 67
3.89 Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for eLU Activation 67
3.90 Confusion matrix for Softplus Activation and SGD optimizer for LSTM 68
3.91 Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for Softplus Activation 68
3.92 Confusion matrix for Sigmoid Activation and Adadelta for LSTM . . . 69
3.93 Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for Sigmoid Activation 70

3.94 Confusion matrix for reLU Activation and Adadelta optimizer for LSTM 70
3.95 Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for ReLU Activation 70
3.96 Confusion matrix for eLU Activation and Adadelta optimizer for LSTM 71
3.97 Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for eLU Activation 71
3.98 Confusion matrix for Softplus Activation and Adadelta optimizer for LSTM 71
3.99 Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for Softplus Activation 72

Abstract

In today's world of digitization, access and use of internet has been increased drastically. These raises concern about security of digital information and threat of intrusion or denial of services.

Intrusion detection system can be an installed software or piece of hardware which can assist to detect illegal and malicious traffic which violates the security policy. At present, Hackers uses different techniques to get access to the valuable information. Intrusion detection techniques and methods will help in order to detect and prevent those kind of severe and novel attacks.

Having access to most important data features gives you lot of flexibility when you start applying labels. Auto encoders are an important family of neural networks which are well suited for this task. RNN deep learning model has a simple structure with built in feedback loop, allowing to act as a forecasting engine. CNN has been go to solution for machine vision projects for last few years.

In this project, Stacked Auto encoder, Convolution Neural Networks and Recurrent neural networks are created using keras over Microsoft's cognitive Toolkit which is open-source software library for Artificial Intelligence. The NSL-KDD data set was used as it solved some of inherent problems of the KDD'99 data set. In this project, different activation functions, optimizers and variable batch size were used for learning parameters along with greedy layer training technique for our models.

Chapter 1

Introduction

With ever increasing use of Computers and devices connected to each other via a network, it is the primary responsibility to take care of one's security and authenticity. The world wide web has evolved due to the interconnection of computer networks across the globe. It could be used for unethical activities such as to get unauthorized access to someone's local network. Availability of computing facilities can also be targeted by Denial of Service (DoS) attacks. There after, due to increase in network attacks from past few years, intrusion detection system have become mandatory addition to the security infrastructure of any organization.

1.1 Intrusion Detection System

Intrusion detection systems can be either software or hardware that could be used to detect suspicious activity at host as well as network level. It could be categorized in to main two categories: Anomaly detection system and Signature based intrusion detection system. Signature based detection can be cheap to deploy but it is easy to evade once it is known to attackers. While on the other hand anomaly detection can detect potentially wide range of attacks but once in a while they may miss the known attacks as they totally depends on quality of training data.

Intrusions are caused by attackers accessing the systems from the Internet, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given to them. Intrusion Detection Systems (IDSs) are either a software or hardware product that automate this monitoring and analysis process.

Intrusion Detection Systems (IDS) has more benefits from a firewall as it monitors and analyze the network traffic and checks it for anomalies and abnormalities in the network. If it finds any abnormal or novel attack then it raises an alert in the system or informs the system administrator of the attacks whether it is originating from outside and also has the ability to check if the system misuse or attacks are originating from inside the organization.

The attacks may be of various types and the system should be able to predict and identify the attacks accurately, without giving false positives, hence the accuracy of the system becomes more important.

1.2 Why should one use intrusion detection system ?

1. To prevent attacks and violations which can not be prevented by any other security means.
2. To prevent unauthorized access and punish those who would either attack or abuse the system.
3. Detect network based attacks
4. This could be useful as quality control for security designs in large organizations.
5. Can become helpful to document the existing threat to an organization.

1.3 Classification of Intrusion Detection System

1.3.1 Host based Intrusion Detection System

Host based Intrusion Detection System (HIDS) runs on individual hosts or devices on the network. A HIDS monitors the incoming and outgoing packets from the device only and will alert the user or administrator of suspicious activity detected. The suspicious activities are based on the type detection technique employed. For example, audit analysis technique is able to identify the activities related to operating-system-level intrusion and application-level intrusions.

1.3.2 Network based Intrusion Detection System

Network Intrusion Detection System (NIDS) are placed at particularly defined places with in the network to monitor the traffic going in and out of all devices in the network.NIDS are mostly passive devices that monitor the on-going network activity without adding significant overhead or interfering with the network operations. They are easy to install and are secure against attack and may even be undetectable to attackers. They also require little effort to install and use on any existing networks. Ideally it would scan all inbound and outbound traffic; however, doing so might create a bottleneck that would impair the overall speed of the network.

1.4 Advantages of Host Based Intrusion Detection System

1. Detects attacks which can not be seen by network based system.
2. Switched networks do not affect the HIDS.
3. Analyses activity on the host.
4. Cost of additional hardware can be saved.
5. Looks at logs and activity to detect anomaly.
6. Verification of either success or failure of attack is possible.

1.5 Advantages of Network Based Intrusion Detection System

1. Few strategically placed NIDS are capable enough to monitor large network.
2. Very useful to monitor traffic in specific segment.
3. Monitors traffic in real time.
4. Quick response to intrusions.
5. Can analyze the traffic in multiple layers of network stack.
6. Cost effective and easier way to deploy.
7. Backend management server can collect information of previous attacks from sensor and information of failed attacks can be obtained.

1.6 Limitations of Intrusion Detection Systems

1. IDS is like a CCTV camera. It can record the attack only after the occurrence but can not do anything to stop the attack.
2. Vulnerability based on network administrator.
3. There might be a case where IDS will tell admin something is wrong in the network without specifying further information.
4. IDS will not be able to protect the network in case of internal user decide to attack.
5. Encrypted packets are not processed by IDS and it would be easy for attacker to gain access via encrypted packets.

6. Attacks over multiple fragment will be impossible to detect by the IDS.

7. In general IDS or NIDS are statistically based instructions which can detect attacks through anomaly. However anomaly can occur due to new activities by user in the network.[4]

Chapter 2

Deep Neural Networks

2.1 Artificial Neural Network

One of the main tools used in Machine learning is Artificial Neural Network. As the middle part of the name suggests "Neural" , which is Human brain inspired system which could be intended to replicate the way we human learns. Basic neural network will have an input layer, one output layer along with a hidden layer consisting of units that process given input into something that the output layer can use. ANN are incredible tools for recognizing patterns which are way too complex for programmers to extract and teach the machine to recognize.

Neural Networks have been in to existence since last century, however it is during last decade they have become a major part of artificial intelligence due to the technique called back propagation.

ANNs, like humans learn by examples. An ANN is configured for a specific task or application. These task varies from patten recognition in computer-aided diagnosis, speech recognition to classification of data such as text into various categories. Today ANNs find its use in variety of applications, Google's search are optimized by these neural networks, Facebook's predictive algorithm in news feeds to Amazon's virtual assistant Alexa.

Neural Network with their ability to derive meaning from complicated and huge data, that would have been otherwise impossible for either humans or other computer techniques, is what makes it powerful tool. Other advantages include Adaptive Learning, Self Organization, and Fault Tolerance.

An artificial neuron is the building block of an ANN. Every input is multiplied with random weights so that the inputs are weighted. These weighted inputs are then summed up with biases and passed through a transfer function and then the net input is passed through the activation function as shown in the Figure 1.

$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + b) \tag{1}$$

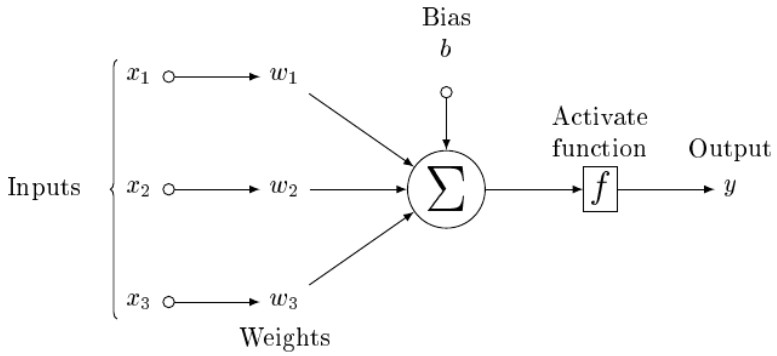

Figure 1: Structure of Artificial Neuron

Although the structure and computation of a single artificial neuron looks simple and easy, but its full potential and power of calculation is realized when they are interconnected and made into an artificial neural network as show in Figure 2

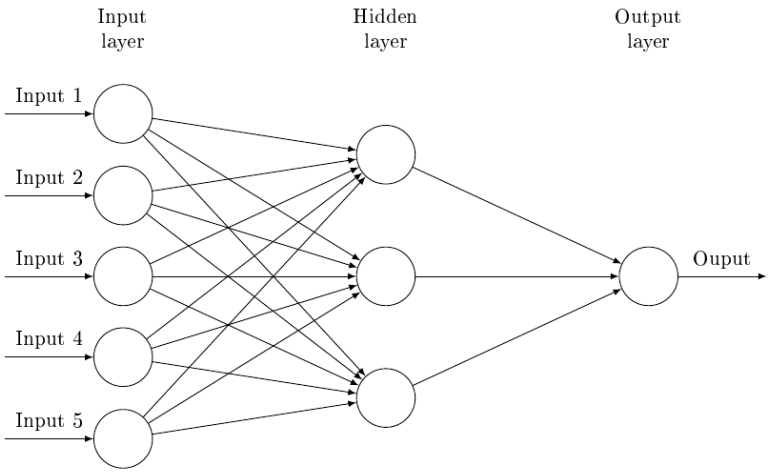

Figure 2: Structure of Artificial Neural Network

These artificial neural networks are interconnections of artificial neurons, however these neurons can not be connected in any way. There are predefined architectures and topologies which make them more beneficial to use. These architectures are very effective in numerous tasks and can solve problems quickly and effectively. Once we know the type of problem we need to solve, then we can select appropriate architecture or topology for the artificial neural network and run our model.

Once the appropriate topology is selected, we are ready to train our model. Just like a human brain which learns from its responses through the inputs from all the sensory organs, for an artificial neural network to do whatever its intended to do it requires training. There are two phases of training, one is pre-training and another is fine tuning phase. As ANN is successful in solving the problems we have trained for, it can be then used for solving the problem we intend to. Artificial neural networks are used in the fields of chemistry, medicine, banking, stock market. It can also solve problems like facial recognition, time-series prediction, regression analysis, pattern recognition.

2.2 Artificial Neural Networks: Types

As we know the building block of Artificial Neural Network (ANN) is the artificial neuron, and when we have more then two artificial neurons working together, we have an artificial neural network. A single neuron is not powerful enough to solve big and complex real world problems, but when they are put together in a specified topology or architecture they can be very handy in solving the same task a single neuron fails to do so. Advantage of artificial neural network is that these neurons can process data in distributed way, in parallel, non-linearly and locally.

The ways in which these neurons are connected is what makes them so powerful. There are basically two main topologies in which these neurons are connected.

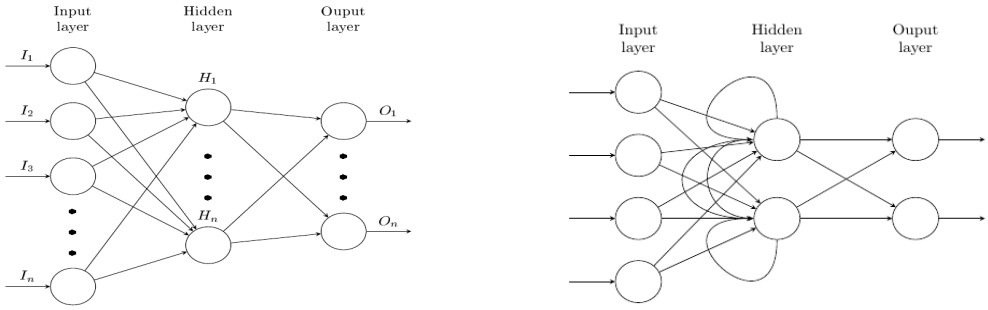

Figure 3: Feed Forward and Recurrent Neural Network

As we can see from the Figure 3 that in feed forward neural network the flow of information is only in one direction that is from inputs to the outputs wherein in the recurrent neural network the information flows in many possible directions as show in the Figure 3

There are other neural networks as follows

- Convolutional Neural Networks
- Long sort-term memory
- Deep Belief Networks
- Stacked Auto-encoders

- Hopfield Neural Networks
- Elman and Jordan Artificial Neural Networks
- Generative Adversarial Networks
- Boltzmann and Restricted Boltzmann Machines

2.3 Training a Deep Neural Network

Training a neural network *Deep or Shallow* requires some parameters to be set. These parameters are important in a way that it can determine the way a neural network will be producing results. Once the structure of the network is decided i.e. how many layers would be needed for that specific task and what will be the number of neurons in each layer, will decide whether a shallow or a deep network is required or not. After the structure of the neural network is decided upon then it has to be decided what type of *activation function* is to be used, what type of learning algorithm to be implemented, how many number of epochs will be required, what kind of learning mechanism will be required, will it be supervised, unsupervised or reinforcement learning. These parameters will decide what kind of output will it be producing.

2.3.1 Supervised Learning

Supervised Learning is similar to how a human brain learns. Inputs and corresponding outputs are given to the network, this makes it easy for the network to classify or recognize the inputs as they are, then a new set of inputs are given to the network which the network has never seen before and its told to recognize based on the previous learning. Initially the weights are randomly selected and the neural network then maps the inputs and compares the corresponding outputs to the given labels or targets. Based on the comparison, an error signal is generated. Ideally the error signal is zero if the outputs are equal or similar to the inputs given, this error signal is then back propagated to the network. System then adjust its weights according to the error signal and optimize the learning to make it more accurate. Repetition of this process will minimizes the error signal to a point where it is within our error tolerance level. Figure 4 shows how the error calculated at the end is back propagated to the network.

Some of the supervised learning algorithms are listed below

- Perceptron Learning
- Back-propagation
- Newton Method
- Grossberg Learning

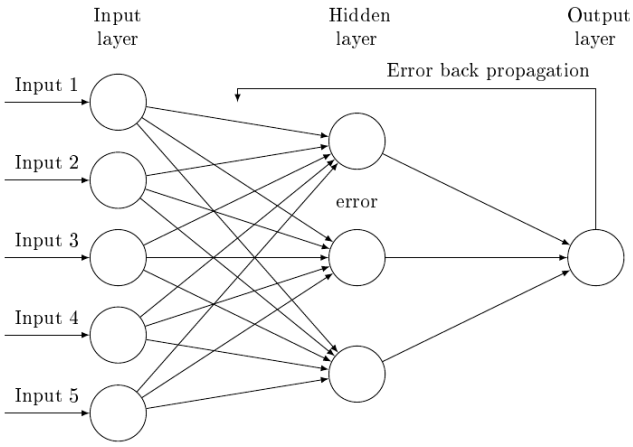

Figure 4: Supervised Learning

2.3.2 Unsupervised Learning

Contrary to supervised learning, in unsupervised learning only inputs are provided and no outputs is provided. Input data is then grouped together and the network is trained on the features decided by the model itself. It learns the hidden structure and representation of the data. Many approaches uses unsupervised learning, some of them include clustering, where dataset given is to be clustered or grouped together based on their similarities. In artificial neural networks Generative adversarial networks (GAN) uses unsupervised learning, implemented by a system of two neural networks competing against each other in a zero-sum game framework[7].

An example of unsupervised learning is clustering, in the Figure 5 *k-means* clustering is seen where $k=2$, hence the data is clustered into two different clusters based on their structure.

Unsupervised learning approaches include

- Clustering
- Generative Adversarial Networks(GAN)
- Hebbian Learning

2.3.3 Reinforcement Learning

Reinforcement learning is inspired by behaviorist psychology. As the name suggest it reinforces itself to maximize the performance of the network. Reinforcement learning is different from other learning in a sense that in reinforcement learning we do not provide pair of input and output data, rather an agent takes an action in an environment which is then interpreted by an interpreter or critic which then rewards the agent accordingly and this feedback is then analyzed and agent improves its performance in accordance to

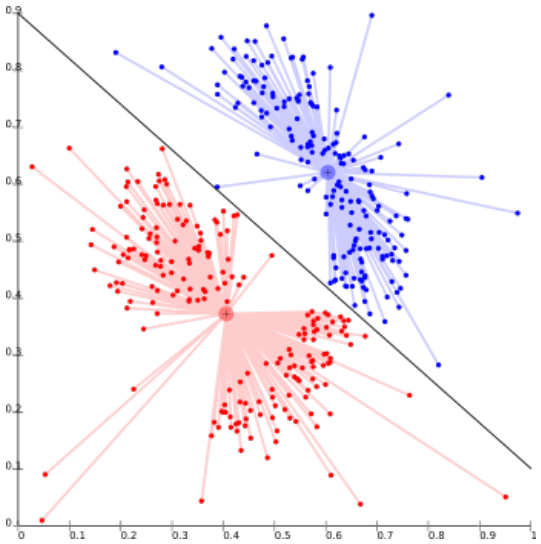

Figure 5: Unsupervised Learning(K-Means)

the reward.

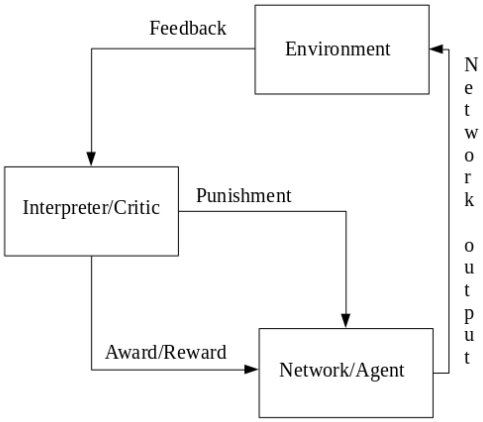

Figure 6: Reinforcement Learning

2.3.4 Activation Functions

Activation function in a neural network can be described as a function which transfers or translates the input signal to output signal. Comparing it to an electrical circuit an activation function is one which can be "*ON*" (1) or "*OFF*" (0) depending on the input it receives. As the artificial neurons are inspired by the biological neuron, the activation function in biological from is either the neuron is firing or not.

Activation functions are selected for a specific application and are highly application dependent. There are many applications which needs classification and there are many application which require prediction, hence the properties of different activation's are used in different applications. The most common activation functions with its properties are listed below. However we have used only four of the activation functions namely *Sigmoid Activation, ReLU activation, eLU activation, Softplus activation* and in all the models the last layer of the model has *Softmax activation*.

Sigmoid Activation Function

Sigmoid is one of the most common activation function used in neural networks. This function is widely used because its easy to calculate their derivatives, which makes weights calculation very easy in some cases. Sigmoid activation has the vanishing gradient problem.

$$S(x) = \frac{1}{1 - e^x} \tag{2}$$

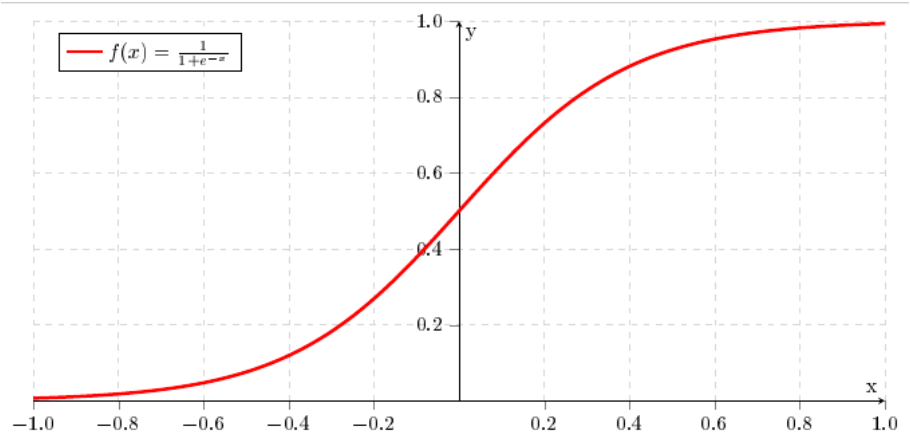

Figure 7: Sigmoid Activation Function

Step Function

Step function was used in perception. The output is certain value, if the sum of the inputs is above or below a certain threshold. These kind of activation functions were used in case of binary classification. In other words, if there are only two classes to classify we can use this activation function.

$$f(x) = \begin{Bmatrix} 1 & x \in A \\ 0 & x \notin A \end{Bmatrix} \tag{3}$$

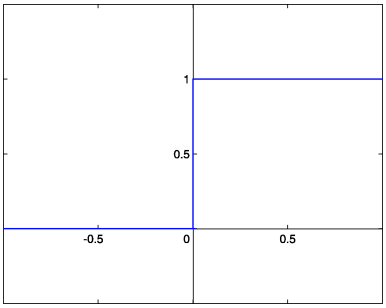

Figure 8: Step Activation Function

Rectified Linear Unit (ReLU)

Rectified Linear Unit function was first introduced by Hahnloser in 2000 with strong biological motivation and justification. This is the most popular activation function from deep neural networks,because unlike Sigmoid and TanH activation ReLU does not have a vanishing gradient problem[5].

$$f(x) = \begin{Bmatrix} 0 & x < 0 \\ x & x \geq 0 \end{Bmatrix} \tag{4}$$

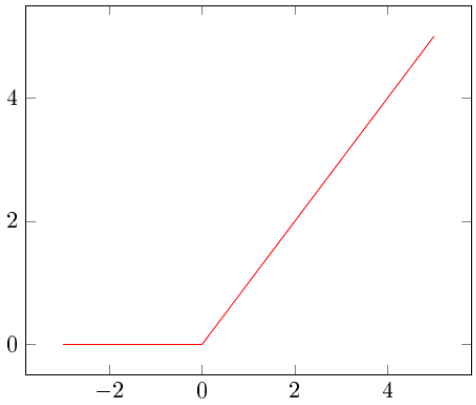

Figure 9: Rectified Linear Unit Activation Function

TanH Activation Function

TanH activation function has characteristics similar to sigmoid activation function, its a scaled sigmoid. It is nonlinear in nature, and its gradient is stronger than sigmoid. Deciding on which to use will depend on the application. Like sigmoid, TanH also has the vanishing gradient problem.

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5}$$

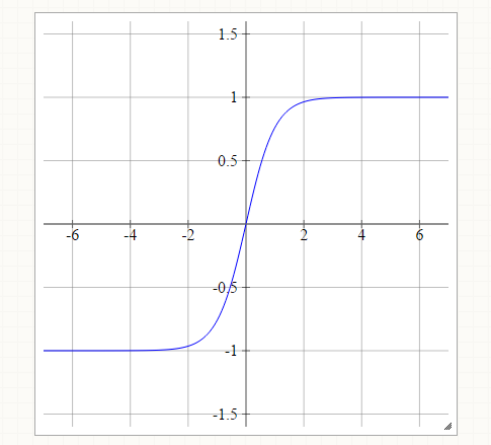

Figure 10: TanH Activation Function

Exponential Linear Unit (eLU)

Exponential Linear Unit alleviate the vanishing gradient problem via the identity for positive values. However, Exponential Linear Units have improved learning characteristics compared to the units with other activation functions [6].

$$f(\alpha, x) = \begin{Bmatrix} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{Bmatrix} \tag{6}$$

Softmax Activation Function

Softmax activation function is a generalization of the logistic function, in probability theory the output of the softmax function can be used to represent a categorical distribution, that is a probability distribution over *X* possible outcomes. Hence, softmax activation function is used in various multi-class classification in artificial neural networks.

$$f_i(\overrightarrow{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_i}} \forall i{=}1...J \tag{7}$$

Softplus Activation Function

Softplus activation function is considered to be "smooth version of ReLU". Softplus has nonzero derivative everywhere on the graph which is contrast to ReLU.

$$f(x) = ln(1 + e^x) \tag{8}$$

z

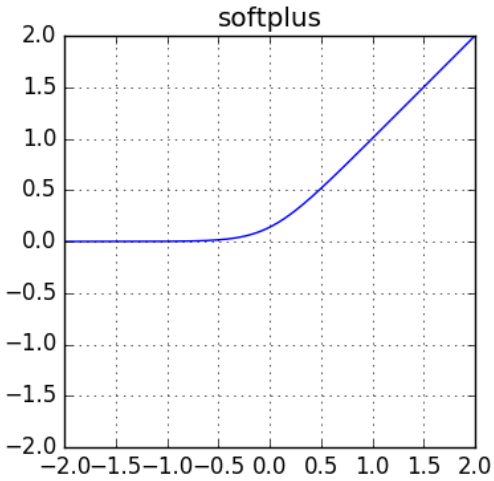

Figure 11: Softplus Activation Function

2.3.5 Optimization Algorithm

Optimization algorithms in neural networks helps in minimizing or maximize a loss function which is simply a mathematical function dependent on models internal parameter which calculates models inability in computing the target values from the set of predictions. The internal parameters such as weights and biases values of a neural network which plays a major role in training process of the neural network, optimizing algorithm helps in setting the values such that there is practically no difference between the predicted value and the value that was feed to the network.

In this project of ours we have used four different optimizers to minimize the loss function, namely *Adam Optimizer, Stochastic Gradient Descent optimizer, RMSProp Optimizer, Adadelta Optimizer*

Stochastic Gradient Descent Optimizer

Gradient descent is also called steepest descent. It is used to find the minimum of a function. It is also the popular method in the field of machine learning to find the highest accuracy and to minimize the error rate of a given training set and is used to calculate the minimum error by minimizing the cost function . To find the gradient descent, we need to take some steps which are proportional to the negative of the gradient of the

function at a point. And if we take steps which are proportional to the positive gradient it reaches the local maximum of that particular function. This is known as gradient ascent[9].

$$w = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n \tag{9}$$

Where:

w : is the parameter to be estimated
$Q_i(w)$: is is the value of the loss function at the i-th example
η : is the step size (sometimes called the learning rate in machine learning)

Adam Optimizer

ADAM (Adaptive Moment Optimizer) is an extension to Gradient Descent optimizer which is seen these days as a broader option for deep learning applications in case of computer vision and natural language processing. It is used instead of GD to update the weights in the network based on iterations in the training data and is used to calculate the adaptive learning rate for each parameters (weight and bias)[9]. It's an popular algorithm in deep learning and have shown good results combining the properties of both AdaGrad and RMSProp algorithms which could handle the noise problems

Adadelta Optimizer

Adadelta is a novel approach which uses per-dimension learning rate method for gradient descent. This method dynamically adapts over time using only first order information/values. Adadelta unlike adagrad which computes the gradients over the whole past epochs, this one accumulates the gradients values for fixed window of epoch. It averages the gradients value which were from selected from the window, so that during optimization it has to look only for past average value and current gradient value.
[21]

RMSProp Optimizer

It is able to increase or decrease the learning rate which the other optimizer named Addgradis not able to. In RMSProp overall learning rate will be divided by the square root of the sum of squares of the previous update gradients for a given parameter. The difference is that RMSProp doesn't weight all of the previous update gradients equally, it uses an exponentially weighted moving average of the previous update gradients which means that the old error values doesn't contribute much [9]. In this way this optimizer jumps around the optimum.

2.4 Challenges in Training a Deep Neural Network

Deep Neural Networks are comprised of multiple hidden layers, so many issues can arise while training a DNN. Three most common issues are *Overfitting*, *Computation time Underfitting.*

DNNs are prone to overfitting because of the added layers of abstraction, which allows them to model rare dependencies in the training data. Regularization methods such as weigh decay (l2- regularization) or sparsity (l1 regularization) can be applied during the training of the model to combat overfitting[12].

Training a DNN with multiple hidden layers consisting of hundreds of thousands of neurons can be a lengthy and time consuming process. Considering many training parameters such as size, the learning rate, initial weights,biases and trying each and every possible combination can be a time consuming process and may not even be a feasible process[13]. Various methods such as batch size, parallel processing and distributed computing can speed up the computation process. Use of GPU's can also speed up the process.

2.5 Stacked Autoencoder

An autoencoder is an simple artificial neural network first introduced in 1980 by Hilton and the PDP group [14]. Autoencoders are used for unsupervised learning. They were introduced to address the problem of "backpropagation without teacher", by using inputs as teacher [15]. The aim of an autoencoder is to learn a representation of a data. Recently, the autoencoders have become widely popular and used for learning generative models in unsupervised pre-training.

2.5.1 Structure

In its simplest form an autoencoder is a feedforward, non recurrent model. It can be generalized as a network having input layer, output layer and hidden layer. A stacked autoencoder is nothing but a network we get by stacking hidden layers onto one another and making it a deep architecture. The output layer has same number of nodes as the input layer with the purpose of reconstruction of the inputs. In our case we had to classify the attacks in a network into five classes hence there will be five neurons in the output layer.

In simplest of the cases, where there is only one hidden layer the mathematical representation of autoencoder will be,

$$y = \sum_{i=0}^{n} \sigma(W_i x_i + b_i) \tag{10}$$

Where:

y : is the output of the network
i : is the number of neurons in a layer
σ : is the activation or the transfer function
W : is the weights vector
x : is the values of input signal
b : is the values of bias added

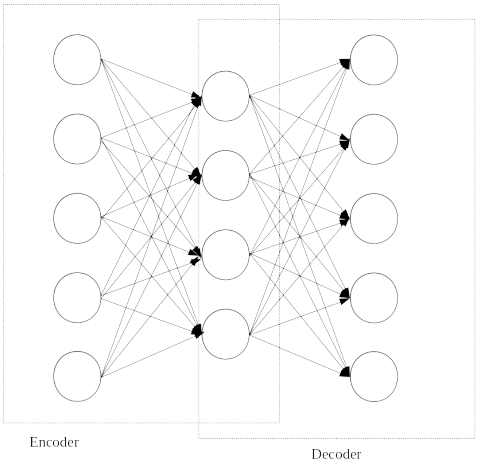

Figure 12: Autoencoder with single hidden layer

2.5.2 Types of Autoencoders

The simple autoencoder have some disadvantages, in some cases it was unable to learn the important features effectively. Hence, there was a need for improving the ability of autoencoders to learn the important features and representations for highly accurate results, and hence there was need of Denoising and Sparse autoencoders .

Denoising Autoencoders

When an autoencoder is trained, its necessary that the inputs provided for training purpose are not corrupted. After the training is done and the model is applied to a real world problem there is a possibility that the inputs given to autoencoder are corrupted, this may destabilize the model, and the results may not be robust. In order to tackle this problem, in *Denoising autoencoder* takes partially corrupted inputs in the training phase to recover the uncorrupted inputs.

Sparse Autoencoder

In *Sparse autoencoder*, we have large number of hidden units, more than inputs, by doing so the autoencoder can learn more useful structures and representation of the data. This kind of autoencoder is useful in classification tasks. Also sparsity can be achieved by tweaking the loss function during training (comparing probability distribution of the hidden unit activation with some low desired values) or by manually choosing the strongest hidden activation functions and ignoring the rest refereed to as *k-sparse autoencoders*[16].

The difference between Stacked autoencoder, Denoising autoencoder and Sparse autoencoder is that when we stack number of layers of encoder and similarly the decoders what we get is Stacked Autoencoders. This stacking helps in learning robustly the underlying structure in the data, Denoising Autoencoders are different in a way that it takes inputs with noise which makes it robust for any real world data and less susceptible to instability. Sparse autoencoders like stacked autoencoder also helps in understanding the underlying structure of the input data.

2.6 Convolutional Neural Networks

Convolutional Neural Networks(CNN's) are special kind of neural networks which are used to process data that is known to have grid like structure. Unlike other neural networks, CNN's perform *convolution* mathematical operation instead of general matrix multiplications. CNN's now a days are very effectively used for image data, it can also be employed for time series data.

The architecture of Convolutional Neural Network is shown below.

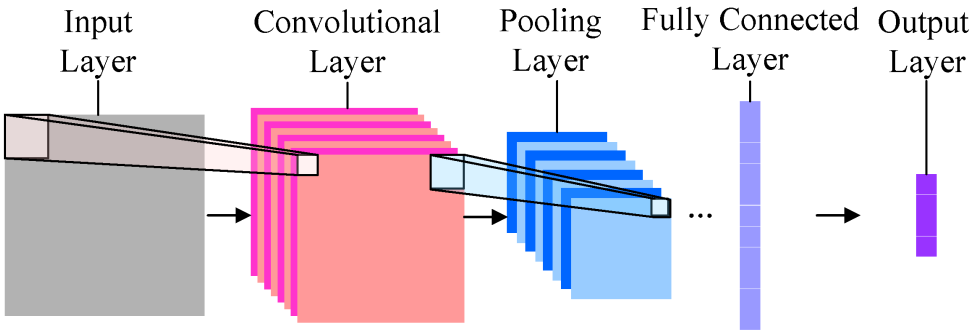

Figure 13: Architecture of Convolutional Neural Network

2.6.1 Convolution

Convolution operation is preformed on input to extract features. CNN has Kernel which performs this operation on input. Mathematically this convolution operation with x as

input and *w* as Kernel at time *t* is represented as

$$s(t) = (x * w)(t) \tag{11}$$

The kernel(w) needs to be a valid probability density function otherwise the output will not be weighted average. The convolutions are used more often in more than two axis at a time. So for an two dimensional image *I* we will use Kernel and the above equation becomes

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, i-n) \tag{12}$$

In a typical layer of convolutional network consist of three stages, in the first stage the layer performs several convolutions in parallel to produce set of linear activation, in the second stage each linear activation is passes through non linear activation such as ReLU, and in the third stage we perform pooling function.
The second stage of passing each produced set of linear activation is necessary as most of the real world data is non linear in nature. ReLU is most often used as it will substitute all the negative values to zeros. There are other non linear functions like tanh and sigmoid.

2.6.2 Pooling

The pooling layer or pooling function reduces the dimensionality, also called subsampling of each output but retains the most important information. So pooling makes representation approximately invariant to small translations of the input, it means that if we have a small change in the input the values of most of the pooled output do not change.
There are two type of pooling functions, *Max Pooling* and *Average Pooling*, In *Max Pooling* the largest element from the rectified output that falls under the kernel area and in *Average Pooling* we take average of the elements that fall under the kernel.

2.7 Recurrent Neural Networks

Recurrent Neural Networks(RNN's) are special type of artificial neural networks they are designed to recognize patterns in sequences of data. They are specially useful for data where the current data point has a relationship with what appeared before it. Sequential data such as text, speech, stock market data etc.

Unlike feed forward networks like Convolutional Neural Networks or Autoencoders, Recurrent networks takes their input not just the current example but what it has seen or perceived previously in time. So the decision a recurrent network reach at t-1 time step affects the decision it will reach a moment later at time step t. It is finding the correlations between events separated by many moments and these correlations are called as "long term dependencies". This sequential information is preserved in the recurrent neural network's hidden state which may span many time steps.

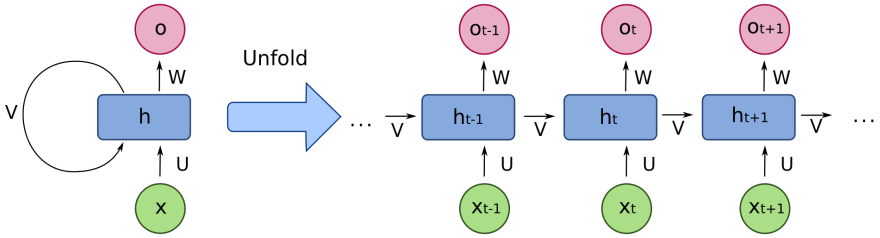

Figure 14: Unfolded Recurrent Neural Network

The mathematical equation of carrying memory forward is:

$$h_t = \phi(W.x_t + Uh_{t-1}) \tag{13}$$

where:

$$\begin{aligned} h_t &= \text{hidden state at time t} \\ \phi &= \text{logistic sigmoid} \\ W &= \text{weight matrix} \\ x_t &= \text{input at time t} \\ U &= \text{hidden to hidden state matrix} \\ h_{t-1} &= \text{hidden state at time t-1} \end{aligned}$$

Due to the long term dependencies RNN's suffer from exploding or vanishing gradient. So if the sequence are quite long which makes the gradient vanish, that is, it becomes too small that it is useless or it explodes that is becomes too big. To tackle this problem of vanishing and exploding gradients a new architecture was introduced called as Long Short Term Memory(LSTM).

2.7.1 Long Short Term Memory

Long Short Term Memory(LSTM) are special kind of RNNs which are capable of learning the long term dependencies. So remembering long sequence is their default behaviour. LSTMs are different then RNNs in a way that LSTM's have Gates. These gates gives LSTM the ability to remove or add information to cell state by regulating these gates. There are three gates in LSTM, in the first step LSTM have to decide what information to keep and what to throw away. This decision is made by the first gate(Sigmoid Layer) called 'Forget Gate'. So it looks at ht-1(previous time step) and xt(input) and outputs 0 or 1 in each cell state. 0 means 'throw away' and 1 means 'keep all' So the equation for this will be *ft*(what to forget) is defined as

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{14}$$

where:

$$\begin{aligned} f_t &= \text{what to forget at time t} \\ \sigma &= \text{sigmoid function} \\ W_f &= \text{weight matrix} \\ x_t &= \text{input at time t} \\ h_{t-1} &= \text{hidden state at time t-1} \\ b_f &= \text{bias matrix} \end{aligned}$$

Next is what information to store in the cell state. This is again divided into two parts, sigmoid layer called 'input gate layer' decides which values to be updated and tanh layer create new vector that could be added to the cell state, next step will be to combine these two and update the cell state.
The mathematical equation for input gate is:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{15}$$

where:

$$\begin{aligned} i_t &= \text{input at time t} \\ \sigma &= \text{sigmoid function} \\ W_i &= \text{weight matrix} \\ x_t &= \text{input at time t} \\ h_{t-1} &= \text{hidden state at time t-1} \\ b_i &= \text{bias matrix} \end{aligned}$$

The mathematical equation for newly created vector by tanh is:

$$\tilde{C} = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{16}$$

where:

$$\begin{aligned} \tilde{C} &= \text{new vector created by tanh} \\ \sigma &= \text{sigmoid function} \\ W_C &= \text{weight matrix} \\ x_t &= \text{input at time t} \\ h_{t-1} &= \text{hidden state at time t-1} \\ b_C &= \text{bias matrix} \end{aligned}$$

The old state is then updated from Ct-1 to Ct, by multiplying it with ft(whatever needs to be forgotten) and then add i_t(input gate) and $\tilde{C}$
So the new cell state becomes:

$$C_t = (f_t * C_{t-1} + i_t * \tilde{C}\ ) \tag{17}$$

where:

C_t = new cell state
f_t = forget gate functionC_{t-1} = previous cell state
i_t = input function at time t
$\tilde{C}$ = new vector created by tanh

The output will be based on our cell state, first sigmoid layer decides which part of cell state to output and then pass that to tanh(push values between -1 and 1) and multiply it with sigmoid again to output it.

$$o_t = sigmoid(W_t[h_{t-1}, x_t] + b_o) \tag{18}$$

where:

o_t = output at time t
W_t = weight matrix
x_t = input at time t
h_{t-1} = hidden state at time t-1
b_o = bias matrix

The hidden state will be:

$$h_t = o_t * tanh(C_t) \tag{19}$$

2.8 Greedy Layer Wise Pre-training

As we have seen already seen in section 2.4 that training a deep architecture of neural network is challenging. So the question remains how can we train deep networks? Geoffrey Hilton proposed greedy layer-wise training algorithm to train Deep Belief Networks [20]. The process involves is to train the layers of a network one at a time, suppose we have a deep neural network of four layers, so first we train a network with one hidden layer and after the first layer is trained we save the weights and biases. After the training is done we add a new layer and train again, in the end we save all the weights and biases, then we train the whole neural network with the weight and biases initialized from the saved ones from pre-training of each layer.

Chapter 3

Evaluation and Results

3.1 The Approach

3.1.1 Pre-Processing

NSL KDD features: NSL KDD being recommended as one of the current best network data comes with 41 features. These features come in mixed values i.e. the features consists of nominal, categorical and continuous values.

Dataset file description

- KDDTrain.ARFF: The full NSL-KDD train set with binary labels in ARFF format
- KDDTrain.TXT: The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
- KDDTrain+_20Percent.ARFF: A 20% subset of the KDDTrain.arff file
- KDDTrain_20Percent.TXT: A 20% subset of the KDDTrain+.txt file
- KDDTest.ARFF: The full NSL-KDD test set with binary labels in ARFF format
- KDDTest+.TXT: The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
- KDDTest-21.ARFF: A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
- KDDTest-21.TXT: A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

Each record from the dataset is labeled with either an attack or labeled as normal.

Attack_in_dataset	Attack_type
DoS	Back,Land,Neptune
Probe	Satan,IPsweep,Nmap,Portsweep,Mscan,Saint
R2L	Guess_password,Ftp_write,Imap,Phf,Multi hop,Warezmaster
U2R	Buffer_overflow

Feature description: NSL KDD comes with 3 types of feature:

- Categorical: These features consist of data with alphabetic values.
- Continuous: These features consists data with values in continuous format.
- Binary: These are the features with values 0 and 1.

Feature_type	Column
Categorical	Protocol_type, flag and service
Continuous/binary	Rest of the 38 features

Data preprocessing: Having different types of features in dataset, we need to preprocess data before giving it to model for train and test purpose.

Categorical data: Categorical data were first label encoded so that the alphabetical values can be converted to numeric values for further processing.

Min-Max normalization: After converting categorical values to numeric, now the whole train and test data is normalized between 0 and 1.

$$X_{new} = \frac{X - min\,(X)}{max\,(X) - min\,(X)} \tag{20}$$

We have considered 3 cases of pre-processing to be used for the CNN model.

- Using normalized values for the CNN as input
- Discretise each feature in 10 bins.
- Convert each 8-bit from discretised values to grayscale value.

Feature Selection

Before giving values to model, we have calculated variance of each feature. We have set threshold of 20% variance that a feature should meet. There was one feature"outbound_cmd" which is having only value '0' throughout the dataset. We removed that feature from the dataset because it will not likely participate in predictions because value is not good for making an instance of a class unique and distinct from another class.

If we set the threshold 40% then we are only left with 16 features which are having values that meet the threshold. It is up to developer on how he/she wants to use the features in what condition. We have set threshold to 20% in entire project.

Discretization: Each feature from 40 feature set in discretized on 10 bins. Previously 40 dimensional vector is now 400 dimensional binary vector after discretization. We wanted to check if we increase the dimension and present it model, will it make any difference from other approach or not?

Greyscale conversion: The 400 dimesional vector after discretization is now reshaped to 50 rows and 8 columns so that we can perform every 8 bit vector to 1 bit grayscale pixel. The output of conversion is a vector of 50 dimensions where each value is grayscale pixel value.

3.1.2 Parameters

Learning Rate

Learning rate is a training parameter that decides how quickly the weight and biases of the neural network are updated during the learning process. This factor is as important as other because if the learning rate is too high then the problem of *overfitting* arises and if the learning rate is too low then it would not reach the local minima for the decided number of iterations. There is no definitive way to know what learning rate should be used, but to do multiple iterations and get the idea of what the learning rate should be.

Epochs

One *Epoch* is the one forward and one backward pass of **all** the training samples in a neural network. Suppose we have 10,000 training samples, than one epoch will be feed pass of all the 10,000 training samples and then one backward pass of the same 10,000 training samples, although these samples will be divided into batches.

Batch Size

It is the number of training examples in one forward and backward pass. Here test set in NSL-KDD dataset has 22,542 records with 41 features, the representation of which

is [22542,41], the batch size we has is *10* so this means it will take [10,41] matrix and feed it to the neural network for forward and backward pass 2254 times in one epoch.

The following attributes were evaluated for all both the models.

Accuracy

Accuracy is the percentage measure of correctly classified or identified records over the total number of records.

Precision (P)

Referred as Positive Predictive Value (PPV) defined as the percentage ratio of the number of *True Positive (TP)* records divided by the sum of *True Positive* and *False Positive (FP)* classified records.

$$Precision = \frac{TP}{TP + FP} \tag{21}$$

Recall (R)

Referred as the TP rate as sensitivity defined as the percentage ratio of number of *True Positive (TP)* records divided by the sum of *True Positive (TP)* and *False Negative (FN)* classified records

$$Recall = \frac{TP}{TP + FN} \tag{22}$$

F-Measure (F1-score)

A measure to represent test accuracy defined as the harmonic mean of precision and recall and represents a balance between them.

$$F - Measure = 2 * \frac{P * R}{(P + R)} \tag{23}$$

Testing and evaluating the model is performed with some fixed settings. The number of epochs taken are *80*, the learning rate of *0.001* and batch size of *10*.

3.2 Data Set Evaluation

There are 3 datasets available to us in NSL-KDD dataset. A generalized neural network need *train data* for training a neural network, *test data* to evaluate the model with different parameters and a *validation set* to validate the training of a model. Here we have all three dataset with their labels as well.

However, the distribution of the attack types in each are different. In test data set there are total of 22,542 records and in train data set there are 1,25,972 records. The distribution of attack types in both test and train set is shown in 3.1

Attack Types	**Number of records in Train Set**	**Number of records in Test Set**
Normal	67342	10003
DoS	45927	7164
Probe	11656	2421
U2R	995	2887
R2L	52	67

Table 3.1: Distribution of records in Test and Train Set

The dataset used for traning is *Test Dataset* and the dataset used for testing is *Train Dataset.*

3.3 Stacked Autoencoder

3.3.1 Results

Implementation of stacked autoencoders was done in Greedy Layer wise pre-training method as explained in section 2.8.
The settings used to evaluate this model are depicted below, if due to some reason the settings are tweaked then it will be mentioned beforehand.

Parameters	**Values**
Learning Rate	0.1
Pre-Training Epochs	10
Fine Tuning Epochs	10
Batch Size	100

Table 3.2: Configuration of Autoencoder for Greedy Layer-Wise Pre-Training

3.3.2 Adam Optimizer

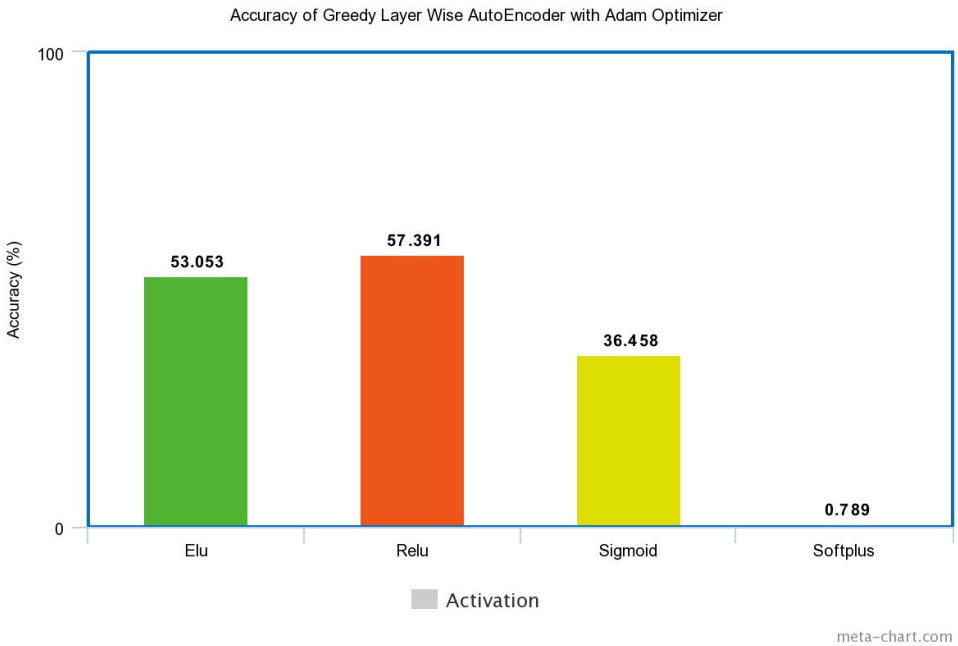

Figure 15: Accuracy of Adam Optimizer for different activation functions

Figure 15 shows that non of the activation behaved as they should have, We have also analyzed all the attack types correctly predicted by different activation in there respective Confusion matrices in the following sections.

The *Precision, Recall and F-Measure* scores of the different activation functions with adam optimizer are listed below.

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	99667342	0	0	0
DoS	0	45927	0	0	0
Probe	0	11656	0	0	0
U2r	0	995	0	0	0
R2L	0	52	0	0	0

Table 3.3: Confusion matrix for Sigmoid Activation and Adam optimizer for Autoencoder

Attack Type	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.36	1.00	0.53	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.00	0.00	52
avg/total	0.13	0.36	0.19	125972

Table 3.4: Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	23420	17594	5365	5574	15389
DoS	89	45239	152	0	447
Probe	9	8002	3626	2	17
U2r	6	819	25	4	141
R2L	0	32	5	1	14

Table 3.5: Confusion matrix for ReLU Activation and Adam optimizer for Autoencoder

Attack Type	Precision	Recall	F1-score	Support
Normal	1.00	0.35	0.53	67342
Dos	0.63	0.99	0.77	45927
Probe	0.40	0.31	0.35	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.27	0.00	52
avg/total	0.80	0.57	0.59	125972

Table 3.6: Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	66729	49	123	3	438
DoS	43623	0	2304	0	0
Probe	11230	6	100	1	316
U2r	978	0	6	0	11
R2L	49	0	0	0	3

Table 3.7: Confusion matrix for eLU Activation and Adam optimizer for Autoencoder

Attack Type	Precision	Recall	F1-score	Support
Normal	0.54	0.99	0.70	67342
Dos	0.00	0.00	0.00	45927
Probe	0.04	0.01	0.01	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.06	0.01	52
avg/total	0.29	0.53	0.38	125972

Table 3.8: Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	0	0	67342	0
DoS	0	0	0	45927	0
Probe	0	0	0	11656	0
U2r	0	0	0	995	0
R2L	0	0	0	52	0

Table 3.9: Confusion matrix for Softplus Activation and Adam optimizer for Autoencoder

Attack Type	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.00	0.00	0.00	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	1.00	0.02	995
R2L	0.00	0.00	0.00	52
avg/total	0.00	0.01	0.00	125972

Table 3.10: Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Softplus Activation

From the above tables, its fairly conclusive that non of the activation functions can be used for Intrusion detection with Adam Optimizer. There are high number of false positives in all the activations for Adam Optimizer.

3.3.3 RMSProp Optimizer

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	67342	0	0	0
DoS	0	45927	0	0	0
Probe	0	11656	0	0	0
U2r	0	995	0	0	0
R2L	0	52	0	0	0

Table 3.11: Confusion matrix for Sigmoid Activation and RMS optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.36	1.00	0.53	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.00	0.00	52
avg/total	0.13	0.36	0.19	125972

Table 3.12: Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	48797	2095	4569	3405	8476
DoS	1638	43710	354	196	29
Probe	1083	120	10354	32	67
U2r	4	2	62	341	586
R2L	4	1	0	4	43

Table 3.13: Confusion matrix for ReLU Activation and RMSProp optimizer for Autoencoder

	precision	recall	F1-score	Support
Normal	0.95	0.72	0.82	67342
Dos	0.95	0.95	0.95	45927
Probe	0.68	0.89	0.77	11656
U2R	0.09	0.34	0.14	995
R2L	0.00	0.83	0.01	52
avg/total	0.92	0.82	0.86	125972

Table 3.14: Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	7497	0	59845	0
DoS	0	7675	0	38222	30
Probe	0	10353	0	1303	0
U2r	0	52	0	943	0
R2L	0	2	0	50	0

Table 3.15: Confusion matrix for eLU Activation and RMS optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.30	0.17	0.21	45927
Probe	0.00	0.00	0.00	11656
U2R	0.01	0.95	0.02	995
R2L	0.00	0.00	0.00	52
avg/total	0.11	0.07	0.08	125972

Table 3.16: Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	46991	3463	3979	12909	0
DoS	1617	43744	499	67	0
Probe	953	665	9390	648	0
U2r	3	1	63	928	0
R2L	0	2	0	50	0

Table 3.17: Confusion matrix for Softplus Activation and RMS optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.95	0.70	0.80	67342
Dos	0.91	0.95	0.93	45927
Probe	0.67	0.81	0.73	11656
U2R	0.06	0.93	0.12	995
R2L	0.00	0.00	0.00	52
avg/total	0.90	0.80	0.84	125972

Table 3.18: Precision, Recall, and F1 Score of Stacked Autoencoder with RMSProp Optimizer for Softplus Activation

ReLU activation for RMSProp outperforms all the other activation functions for this particular case. All the other activation performed worst, one possible reason could be the highly imbalance classes.

3.3.4 Stochastic Gradient Descent Optimizer

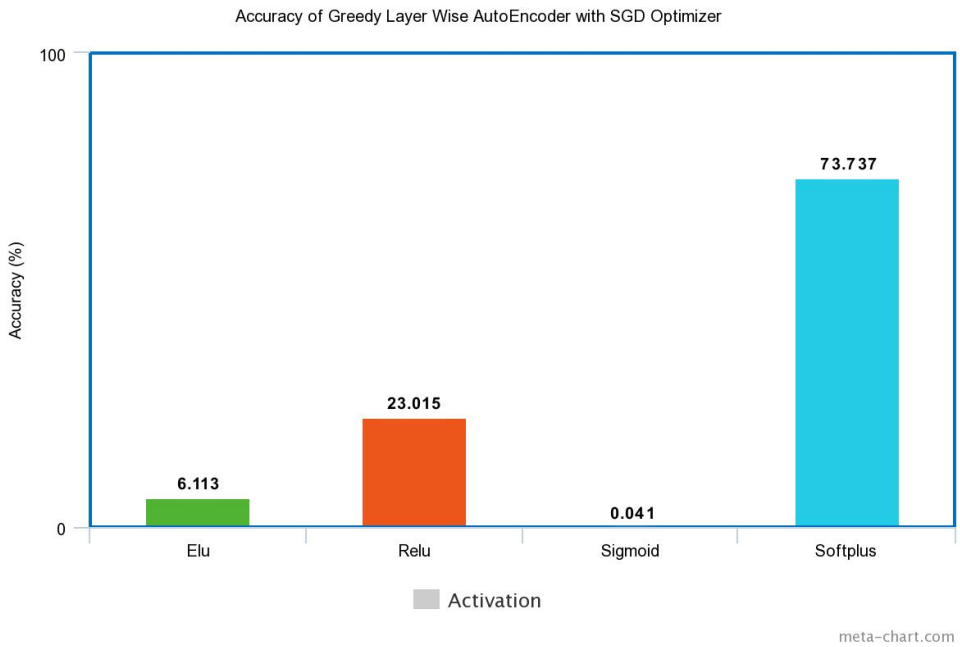

Figure 16: Accuracy of Stochastic Gradient Descent Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	0	0	0	67342
DoS	0	0	0	0	45927
Probe	0	0	0	0	11656
U2r	0	0	0	0	995
R2L	0	0	0	0	52

Table 3.19: Confusion Matrix for Sigmoid Activation and SGD optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.00	0.00	0.00	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	1.00	0.00	52
avg/total	0.00	0.00	0.00	125972

Table 3.20: Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	19	30584	31669	5070	0
DoS	13	24497	2979	18438	0
Probe	4	561	4417	6674	0
U2r	0	362	580	53	0
R2L	0	13	38	1	0

Table 3.21: Confusion matrix for reLU Activation and SGD optimizer for Autoencoder

	precision	recall	F1-score	Support
Normal	0.53	0.00	0.00	67342
Dos	0.44	0.53	0.48	45927
Probe	0.11	0.38	0.17	11656
U2R	0.00	0.05	0.00	995
R2L	0.00	0.00	0.00	52
avg/total	0.45	0.23	0.19	125972

Table 3.22: Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	58	43725	14467	6	9086
DoS	34238	943	8291	1	2454
Probe	251	36	6694	0	4675
U2r	4	361	75	1	554
R2L	0	32	7	0	13

Table 3.23: Confusion Matrix for eLU Activation and SGD optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.02	0.02	0.02	45927
Probe	0.23	0.57	0.33	11656
U2R	0.12	0.00	0.00	995
R2L	0.00	0.25	0.00	52
avg/total	0.03	0.06	0.04	125972

Table 3.24: Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	39968	3073	3685	9688	10928
DoS	525	42186	1792	1288	136
Probe	0	105	10476	951	124
U2r	3	16	47	205	724
R2L	0	1	0	4	47

Table 3.25: Confusion matrix for Softplus Activation and SGD optimizer for Autoencoder

	precision	recall	F1-score	Support
Normal	0.99	0.59	0.74	67342
Dos	0.93	0.92	0.92	45927
Probe	0.65	0.90	0.76	11656
U2R	0.02	0.21	0.03	995
R2L	0.00	0.90	0.01	52
avg/total	0.93	0.74	0.80	125972

Table 3.26: Precision, Recall, and F1 Score of Stacked Autoencoder with Stochastic Gradient Descent Optimizer for Softplus Activation

3.3.5 Adadelta Optimizer

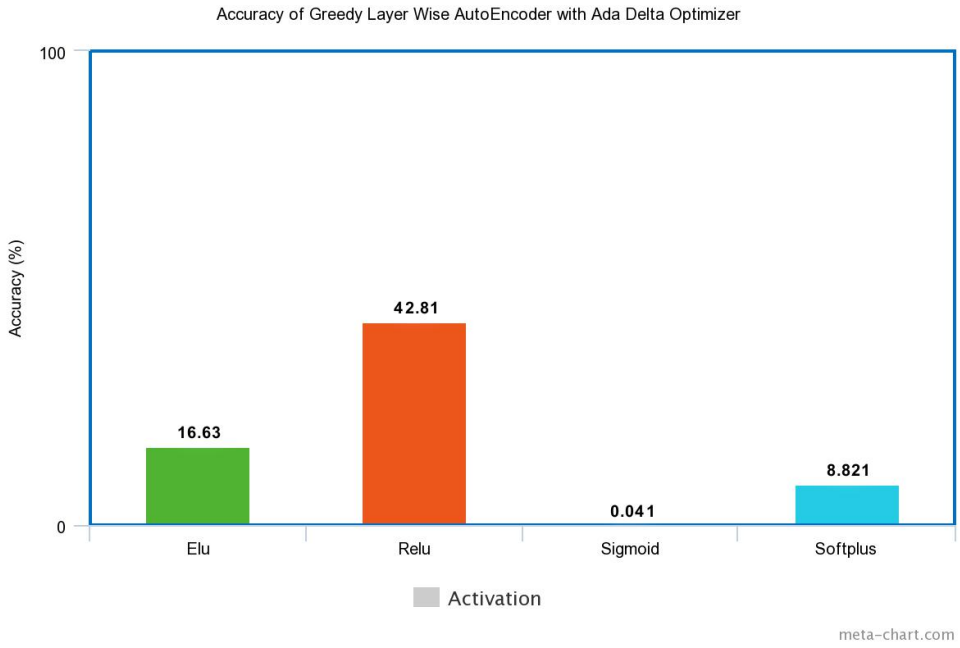

Figure 17: Accuracy of Momentum Optimizer for different activation functions

Unlike Stochastic Gradient Descent Optimizer where Softplus activation was able to *Normal, DoS, Probe, and U2R* attacks, Adadelta Optimizer performed worst for all the activation function we tested.

The *Precision, Recall* and *F-Measure* values for Momentum Optimizer is in the following section.

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	0	0	0	67342
DoS	0	0	0	0	45927
Probe	0	0	0	0	11656
U2r	0	0	0	0	995
R2L	0	0	0	0	52

Table 3.27: Confusion Matrix for Sigmoid Activation and Adadelta optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.00	0.00	0.00	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	1.00	0.00	52
avg/total	0.00	0.00	0.00	125972

Table 3.28: Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	603	16229	17769	8369	24472
DoS	0	44512	598	267	550
Probe	0	3680	7969	5	2
U2r	0	84	63	841	7
R2L	0	16	3	26	7

Table 3.29: Confusion Matrix for reLU Activation and Adadelta optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	1.00	0.01	0.02	67342
Dos	0.69	0.97	0.81	45927
Probe	0.30	0.68	0.42	11656
U2R	0.09	0.85	0.16	995
R2L	0.00	0.13	0.00	52
avg/total	0.81	0.43	0.34	125972

Table 3.30: Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	3114	61503	7	2705	13
DoS	6961	17475	1	21490	0
Probe	5114	1664	329	4548	1
U2r	50	908	0	37	0
R2L	0	49	0	3	0

Table 3.31: Confusion Matrix for eLU Activation and Adadelta optimizer for Autoencoder

	Precision	Recall	F1-score	Support
Normal	0.54	0.99	0.70	67342
Dos	0.00	0.00	0.00	45927
Probe	0.04	0.01	0.01	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.06	0.01	52
avg/total	0.29	0.53	0.38	125972

Table 3.32: Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	63405	1	3936	0
DoS	0	10867	0	35060	0
Probe	0	11253	3	400	0
U2r	0	753	0	242	0
R2L	0	43	0	9	0

Table 3.33: Confusion Matrix for Softplus Activation and Adadelta optimizer for Autoencoder

	precision	recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.13	0.24	0.16	45927
Probe	0.75	0.00	0.00	11656
U2R	0.01	0.24	0.01	995
R2L	0.00	0.00	0.00	52
avg/total	0.12	0.09	0.06	125972

Table 3.34: Precision, Recall, and F1 Score of Stacked Autoencoder with Momentum Optimizer for Softplus Activation

3.4 Convolutional Neural Networks

3.4.1 Adam Optimizer

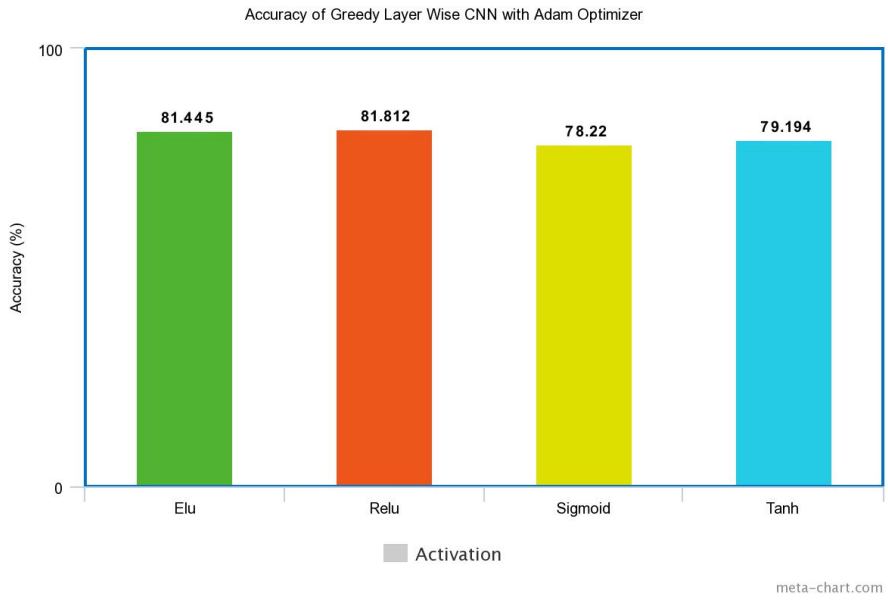

Figure 18: Accuracy of Adam Optimizer for different activation functions

Figure 18 shows that all the activation functions have good accuracy We have also analysed all the attack types correctly predicted by different activation functions as show in table 3.35

The *Precision, Recall and F-Measure* scores of the sigmoid activation with adam optimizer is listed below.

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4692	867	174	0	0
DoS	225	9142	165	0	178
Probe	6	229	845	0	26
U2r	2	1653	58	0	486
R2L	0	16	0	0	21

Table 3.35: Confusion Matrix for Sigmoid activation and Adam Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.95	0.82	0.88	5741
Dos	0.77	0.94	0.85	9710
Probe	0.68	0.76	0.72	1106
U2R	0.00	0.00	0.00	2199
R2L	0.03	0.57	0.06	37
avg/total	0.73	0.78	0.75	18793

Table 3.36: Precision, Recall, and F1 Score of Stacked Autoencoder with Adam Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4770	788	183	0	0
DoS	68	9395	192	49	6
Probe	0	240	866	0	0
U2r	13	1631	224	330	1
R2L	1	18	3	1	14

Table 3.37: Confusion Matrix for ReLU activation and Adam Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.98	0.83	0.90	5741
Dos	0.78	0.97	0.86	9710
Probe	0.59	0.78	0.67	1106
U2R	0.87	0.15	0.26	2199
R2L	0.67	0.38	0.48	37
avg/total	0.84	0.82	0.79	18793

Table 3.38: Precision, Recall, and F1 Score of Stacked Convolutional Neural Network with Adam Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4794	855	74	18	0
DoS	244	9131	228	107	0
Probe	3	285	754	64	0
U2r	20	1388	164	627	0
R2L	0	31	2	4	0

Table 3.39: Confusion Matrix for eLU activation and Adam Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.95	0.84	0.89	5741
Dos	0.78	0.94	0.85	9710
Probe	0.62	0.68	0.65	1106
U2R	0.76	0.29	0.42	2199
R2L	0.00	0.00	0.00	37
avg/total	0.82	0.81	0.80	18793

Table 3.40: Precision, Recall, and F1 Score of Convolutional Neural Network with Adam Optimizer for eLU Activation

TanH Activation

	Normal	Dos	Probe	U2r	R2L
Normal	5011	620	110	0	0
DoS	539	8851	299	18	3
Probe	1	192	911	1	1
U2r	68	1593	430	106	2
R2L	2	25	5	1	4

Table 3.41: Confusion Matrix for TANH activation and Adam Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.89	0.87	0.88	5741
Dos	0.78	0.91	0.84	9710
Probe	0.52	0.82	0.64	1106
U2R	0.84	0.05	0.09	2199
R2L	0.40	0.11	0.17	37
avg/total	0.81	0.79	0.75	18793

Table 3.42: Precision, Recall, and F1 Score of Convolutional Neural Network with Adam Optimizer for TanH Activation

3.4.2 RMSProp Optimizer

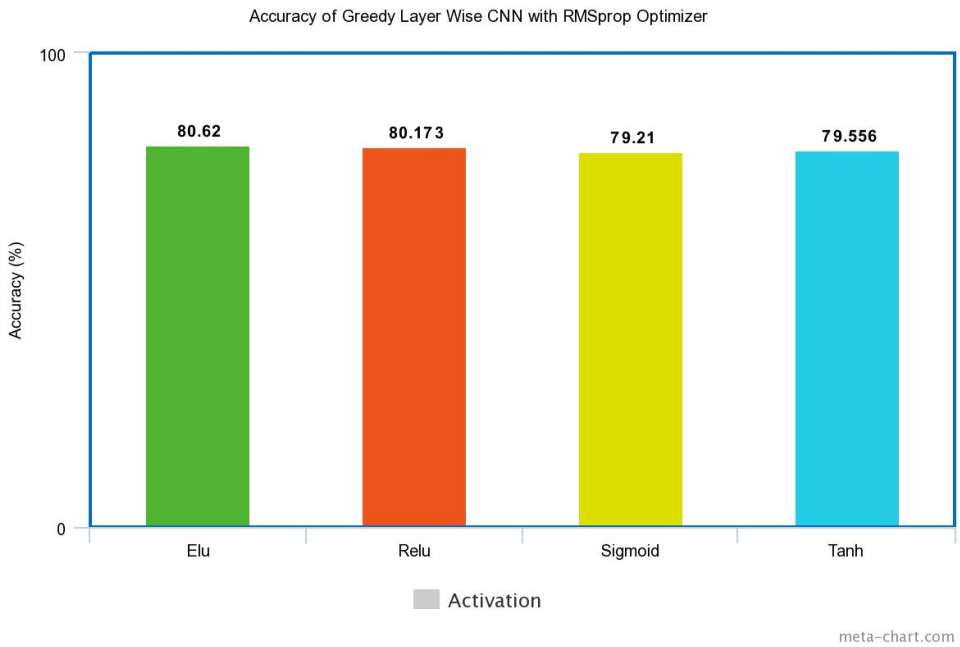

Figure 19: Accuracy of RMSProp Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4621	954	166	0	0
DoS	150	9312	237	11	0
Probe	6	230	870	0	0
U2r	15	1626	475	83	0
R2L	0	31	4	2	0

Table 3.43: Confusion Matrix for Sigmoid activation and RMS Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.96	0.80	0.88	5741
Dos	0.77	0.96	0.85	9710
Probe	0.50	0.79	0.61	1106
U2R	0.86	0.04	0.07	2199
R2L	0.00	0.00	0.00	37
avg/total	0.82	0.79	0.75	18793

Table 3.44: Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	5085	568	88	0	0
DoS	414	9141	149	6	0
Probe	6	275	825	0	0
U2r	6	2131	46	16	0
R2L	0	37	0	0	0

Table 3.45: Confusion Matrix for ReLU activation and RMS Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.92	0.89	0.90	5741
Dos	0.75	0.94	0.84	9710
Probe	0.74	0.75	0.75	1106
U2R	0.73	0.01	0.01	2199
R2L	0.00	0.00	0.00	37
avg/total	0.80	0.80	0.75	18793

Table 3.46: Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	5245	396	92	7	1
DoS	478	8985	230	14	3
Probe	25	251	829	1	0
U2r	40	1792	312	84	1
R2L	0	26	1	2	8

Table 3.47: Confusion Matrix for eLU activation and RMS Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.91	0.91	0.91	5741
Dos	0.79	0.93	0.85	9710
Probe	0.57	0.75	0.65	1106
U2R	0.78	0.04	0.07	2199
R2L	0.62	0.22	0.32	37
avg/total	0.81	0.81	0.76	18793

Table 3.48: Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for eLU Activation

TanH Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4675	1032	34	0	0
DoS	67	9494	143	2	4
Probe	21	320	764	1	0
U2r	2	2033	159	4	0
R2L	0	21	1	1	14

Table 3.49: Confusion Matrix for TanH activation and RMS Optimizer for Convolutional Neural Network

	precision	recall	F1-score	Support
Normal	0.98	0.81	0.89	5741
Dos	0.74	0.98	0.84	9710
Probe	0.69	0.69	0.69	1106
U2R	0.50	0.00	0.00	2199
R2L	0.78	0.38	0.51	37
avg/total	0.78	0.80	0.75	18793

Table 3.50: Precision, Recall, and F1 Score of Convolutional Neural Network with RMSProp Optimizer for TanH Activation

3.4.3 Stochastic Gradient Descent Optimizer

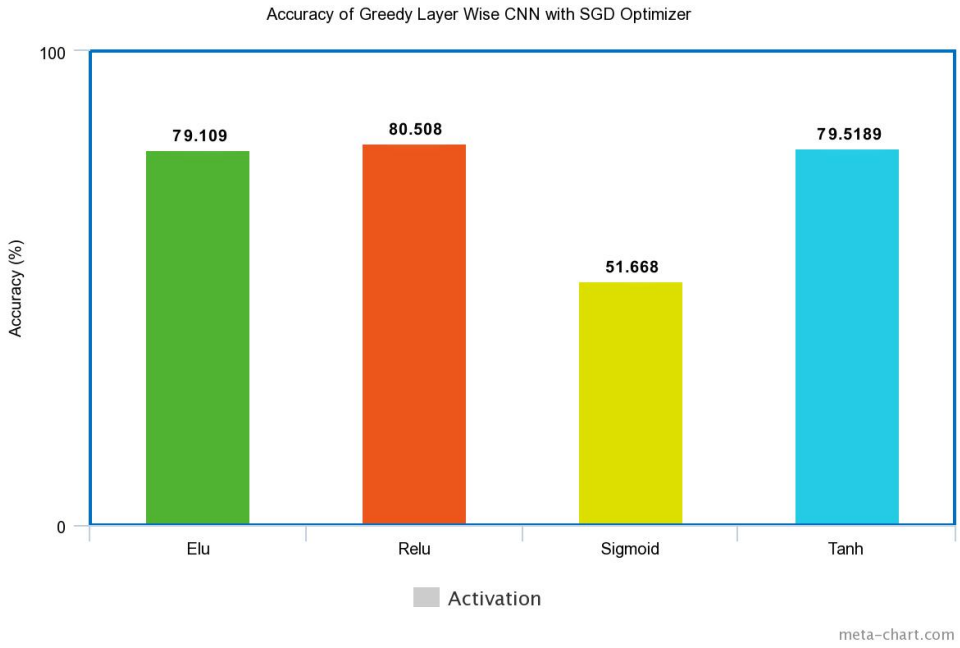

Figure 20: Accuracy of Stochastic Gradient Descent Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	5741	0	0	0
DoS	0	9710	0	0	0
Probe	0	1106	0	0	0
U2r	0	2199	0	0	0
R2L	0	37	0	0	0

Table 3.51: Confusion Matrix for Sigmoid activation and SGD Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	5741
Dos	0.52	1.00	0.68	9710
Probe	0.00	0.00	0.00	1106
U2R	0.00	0.00	0.00	2199
R2L	0.00	0.00	0.00	37
avg/total	0.27	0.52	0.35	18793

Table 3.52: Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4925	726	82	4	4
DoS	360	9049	256	26	19
Probe	0	156	926	24	0
U2r	0	1674	239	219	67
R2L	0	23	2	1	11

Table 3.53: Confusion Matrix for ReLU activation and SGD Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.93	0.86	0.89	5741
Dos	0.78	0.93	0.85	9710
Probe	0.62	0.84	0.71	1106
U2R	0.80	0.10	0.18	2199
R2L	0.11	0.30	0.16	37
avg/total	0.82	0.81	0.77	18793

Table 3.54: Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4848	752	141	0	0
DoS	353	9094	243	6	14
Probe	3	278	825	0	0
U2r	9	1774	357	85	4
R2L	0	16	5	1	15

Table 3.55: Confusion Matrix for eLU activation and SGD Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.93	0.84	0.89	5741
Dos	0.77	0.94	0.84	9710
Probe	0.53	0.75	0.62	1106
U2R	0.92	0.04	0.07	2199
R2L	0.45	0.41	0.43	37
avg/total	0.82	0.79	0.75	18793

Table 3.56: Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for eLU Activation

TanH Activation

	Normal	Dos	Probe	U2r	R2L
Normal	5039	536	164	0	2
DoS	426	8943	321	2	18
Probe	0	220	886	0	0
U2r	0	1749	374	63	13
R2L	0	20	4	0	13

Table 3.57: Confusion Matrix for TanH activation and SGD Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.92	0.88	0.90	5741
Dos	0.78	0.92	0.84	9710
Probe	0.51	0.80	0.62	1106
U2R	0.97	0.03	0.06	2199
R2L	0.28	0.35	0.31	37
avg/total	0.83	0.80	0.75	18793

Table 3.58: Precision, Recall, and F1 Score of Convolutional Neural Network with Stochastic Gradient Descent Optimizer for TanH Activation

3.4.4 Adadelta Optimizer

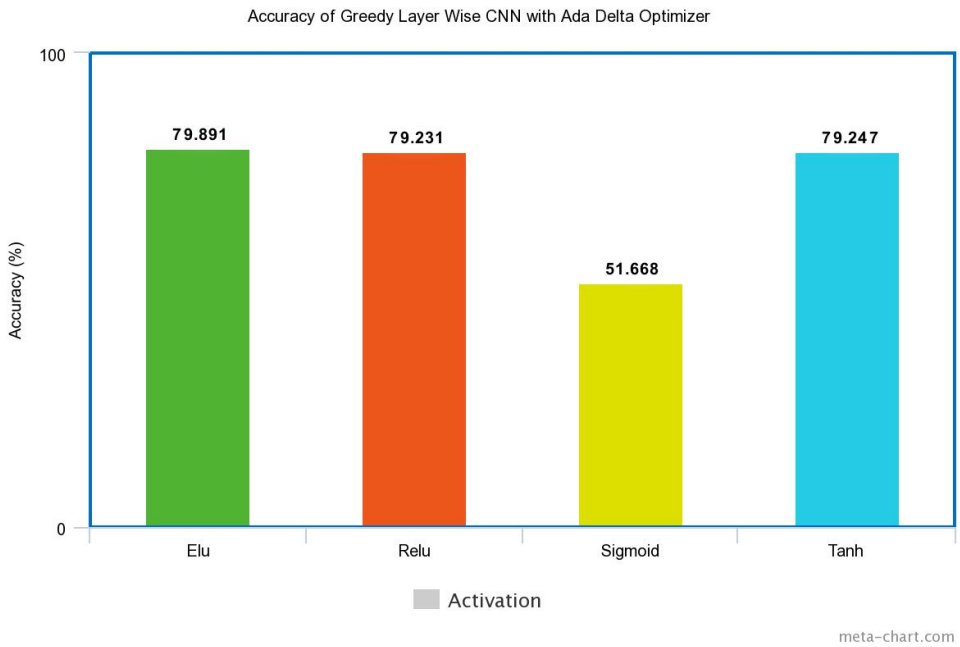

Figure 21: Accuracy of Adadelta Optimizer for different activation functions

The *Precision, Recall* and *F-Measure* values for Adadelta Optimizer is in the following section.

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	5741	0	0	0
DoS	0	9710	0	0	0
Probe	0	1106	0	0	0
U2r	0	2199	0	0	0
R2L	0	37	0	0	0

Table 3.59: Confusion Matrix for Sigmoid activation and Adadelta Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	5741
Dos	0.52	1.00	0.68	9710
Probe	0.00	0.00	0.00	1106
U2R	0.00	0.00	0.00	2199
R2L	0.00	0.00	0.00	37
avg/total	0.27	0.52	0.35	18793

Table 3.60: Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4585	971	185	0	0
DoS	7	9294	409	0	0
Probe	87	8	1011	0	0
U2r	4	1785	410	0	0
R2L	0	33	4	0	0

Table 3.61: Confusion Matrix for ReLU activation and Adadelta Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.98	0.80	0.88	5741
Dos	0.77	0.96	0.85	9710
Probe	0.50	0.91	0.65	1106
U2R	0.00	0.00	0.00	2199
R2L	0.00	0.00	0.00	37
avg/total	0.82	0.79	0.75	18793

Table 3.62: Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation

eLU Activation

	Normal	DoS	Probe	U2r	R2L
Normal	4614	991	136	0	0
DoS	8	9304	398	0	0
Probe	4	6	1096	0	0
U2r	5	1794	400	0	0
R2L	0	33	4	0	0

Table 3.63: Confusion Matrix for eLU activation and Adadelta Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	1.00	0.80	0.89	5741
Dos	0.77	0.96	0.85	9710
Probe	0.54	0.99	0.70	1106
U2R	0.00	0.00	0.00	2199
R2L	0.00	0.00	0.00	37
avg/total	0.73	0.80	0.75	18793

Table 3.64: Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation

TanH Activation

	Normal	Dos	Probe	U2r	R2L
Normal	4653	996	92	0	0
DoS	26	9318	366	0	0
Probe	92	93	921	0	0
U2r	38	1808	352	1	0
R2L	6	24	7	0	0

Table 3.65: Confusion Matrix for TanH activation and Adadelta Optimizer for Convolutional Neural Network

	Precision	Recall	F1-score	Support
Normal	0.97	0.81	0.88	5741
Dos	0.76	0.96	0.85	9710
Probe	0.53	0.83	0.65	1106
U2R	1.00	0.00	0.00	2199
R2L	0.00	0.00	0.00	37
avg/total	0.84	0.79	0.75	18793

Table 3.66: Precision, Recall, and F1 Score of Convolutional Neural Network with Adadelta Optimizer for Sigmoid Activation

3.5 Recurrent Neural Network(LSTM)

3.5.1 Results

The settings used to evaluate this model are depicted below, if due to some reason the settings are tweaked then it will be mentioned beforehand.

Parameters	**Values**
Learning Rate	0.1
Fine Tuning Epochs	10
Batch Size	100

Table 3.67: Configuration of Recurrent Neural Network(LSTM)

3.5.2 Adam Optimizer

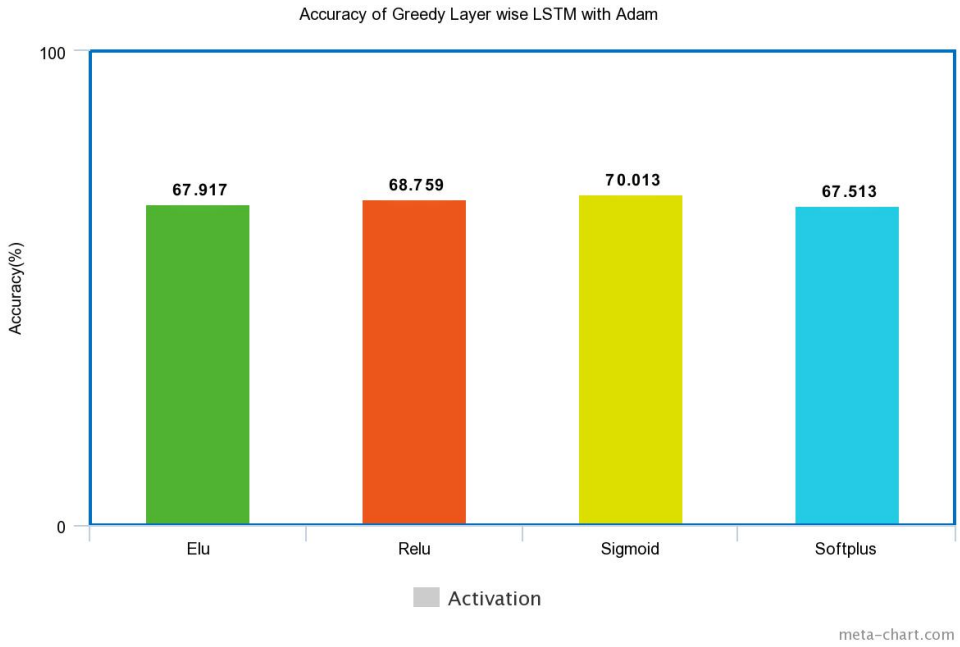

Figure 22: Accuracy of Adam Optimizer for different activation functions

Figure 22 shows that every activation function behaved almost on par with every other activation function we tested.
We have also analyzed all the attack types correctly predicted by different activation in there respective Confusion matrices in the following sections.

The *Precision, Recall and F-Measure* scores of the different activation functions with adam optimizer are also listed below.

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	35939	1570	5483	13052	11298
DoS	921	41685	1517	1451	353
Probe	0	402	10268	623	363
U2r	5	51	15	266	658
R2L	3	0	1	8	40

Table 3.68: Confusion matrix for Sigmoid Activation and Adam optimizer for RNN(LSTM)

	Precision	Recall	F1-score	Support
Normal	0.97	0.53	0.69	67342
Dos	0.95	0.91	0.93	45927
Probe	0.59	0.88	0.71	11656
U2R	0.02	0.27	0.03	995
R2L	0.00	0.77	0.01	52
avg/total	0.92	0.70	0.77	125972

Table 3.69: Precision, Recall, and F1 Score of LSTM with Adam Optimizer for Sigmoid Activation

From the *Confusion Matrix* and from the values of *Precision, Recall and F-Score* we can see that due to high imbalance in the distribution of samples the results of *R2L* attack type are worst, but other than that all the classes were identified as expected.

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	33662	1181	4991	19041	8467
DoS	252	41974	1012	2310	379
Probe	0	433	10170	929	124
U2r	10	2	7	783	143
R2L	1	0	0	22	29

Table 3.70: Confusion matrix for ReLU Activation and Adam optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.99	0.50	0.66	67342
Dos	0.96	0.91	0.94	45927
Probe	0.63	0.87	0.73	11656
U2R	0.03	0.79	0.07	995
R2L	0.00	0.56	0.01	52
avg/total	0.94	0.69	0.77	125972

Table 3.71: Precision, Recall, and F1 Score of LSTM with Adam Optimizer for ReLU Activation

The observations in the seciton 3.5.2 continues to propagate in this section as well due to imbalance in the sample distribution across classes.

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	34119	739	3776	17057	11651
DoS	242	42023	970	2353	339
Probe	8	643	8621	1980	404
U2r	2	1	66	765	161
R2L	1	0	0	22	29

Table 3.72: Confusion matrix for eLU Activation and Adam optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.99	0.51	0.67	67342
Dos	0.97	0.91	0.94	45927
Probe	0.64	0.74	0.69	11656
U2R	0.03	0.77	0.07	995
R2L	0.00	0.56	0.00	52
avg/total	0.94	0.68	0.77	125972

Table 3.73: Precision, Recall, and F1 Score of LSTM with Adam Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	33923	801	2755	22460	7403
DoS	242	41124	1869	2676	16
Probe	7	243	9081	2290	35
U2r	1	54	14	984	32
R2L	1	0	1	24	26

Table 3.74: Confusion matrix for Softplus Activation and Adam optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.99	0.50	0.67	67342
Dos	0.97	0.90	0.93	45927
Probe	0.66	0.78	0.72	11656
U2R	0.03	0.90	0.06	995
R2L	0.00	0.50	0.01	52
avg/total	0.95	0.68	0.76	125972

Table 3.75: Precision, Recall, and F1 Score of LSTM with Adam Optimizer for Softplus Activation

3.5.3 RMSProp Optimizer

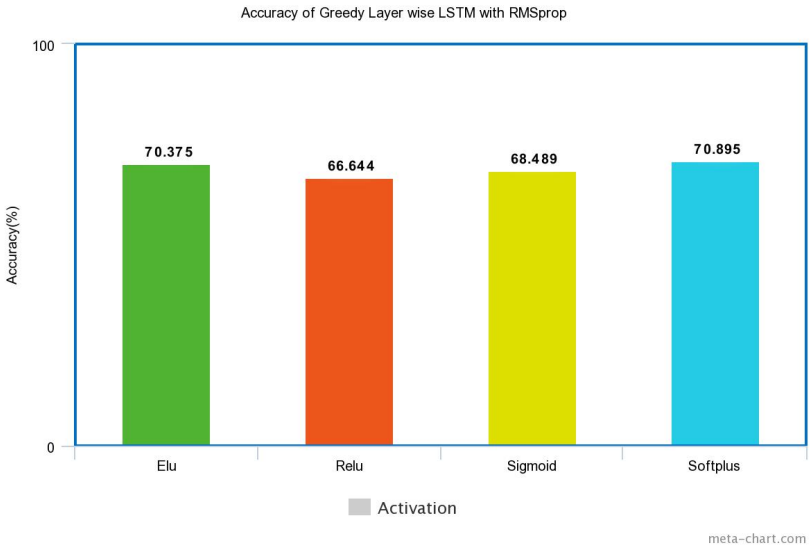

Figure 23: Accuracy of RMSProp Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	34219	1708	2669	20270	7476
DoS	253	42052	947	2480	195
Probe	1	1060	9584	967	44
U2r	499	52	15	396	33
R2L	12	1	0	12	27

Table 3.76: Confusion matrix for Sigmoid Activation and RMSProp optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.98	0.51	0.67	67342
Dos	0.94	0.92	0.93	45927
Probe	0.67	0.82	0.74	11656
U2R	0.02	0.40	0.03	995
R2L	0.00	0.52	0.01	52
avg/total	0.93	0.68	0.76	125972

Table 3.77: Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	34477	1115	3231	20290	8229
DoS	266	39900	1758	3660	343
Probe	18	564	9282	1730	62
U2r	501	1	56	273	164
R2L	21	0	0	10	21

Table 3.78: Confusion matrix for ReLU Activation and RMSProp optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.98	0.51	0.67	67342
Dos	0.96	0.87	0.91	45927
Probe	0.65	0.80	0.71	11656
U2R	0.01	0.27	0.02	995
R2L	0.00	0.40	0.00	52
avg/total	0.93	0.67	0.76	125972

Table 3.79: Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	36031	907	5639	13758	11007
DoS	929	41970	1235	1776	17
Probe	20	360	10245	715	316
U2r	2	52	8	368	565
R2L	4	2	0	7	39

Table 3.80: Confusion matrix for eLU Activation and RMS optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.97	0.54	0.69	67342
Dos	0.97	0.91	0.94	45927
Probe	0.60	0.88	0.71	11656
U2R	0.02	0.37	0.04	995
R2L	0.00	0.75	0.01	52
avg/total	0.93	0.70	0.78	125972

Table 3.81: Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	36244	1751	4413	13460	11474
DoS	1133	42661	321	1553	259
Probe	18	567	10118	635	318
U2r	510	54	11	258	162
R2L	15	2	1	7	27

Table 3.82: Confusion matrix for Softplus Activation and RMS optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.96	0.54	0.69	67342
Dos	0.95	0.93	0.94	45927
Probe	0.68	0.87	0.76	11656
U2R	0.02	0.26	0.03	995
R2L	0.00	0.52	0.00	52
avg/total	0.92	0.71	0.78	125972

Table 3.83: Precision, Recall, and F1 Score of LSTM with RMSProp Optimizer for Softplus Activation

3.5.4 Stochastic Gradient Descent Optimizer

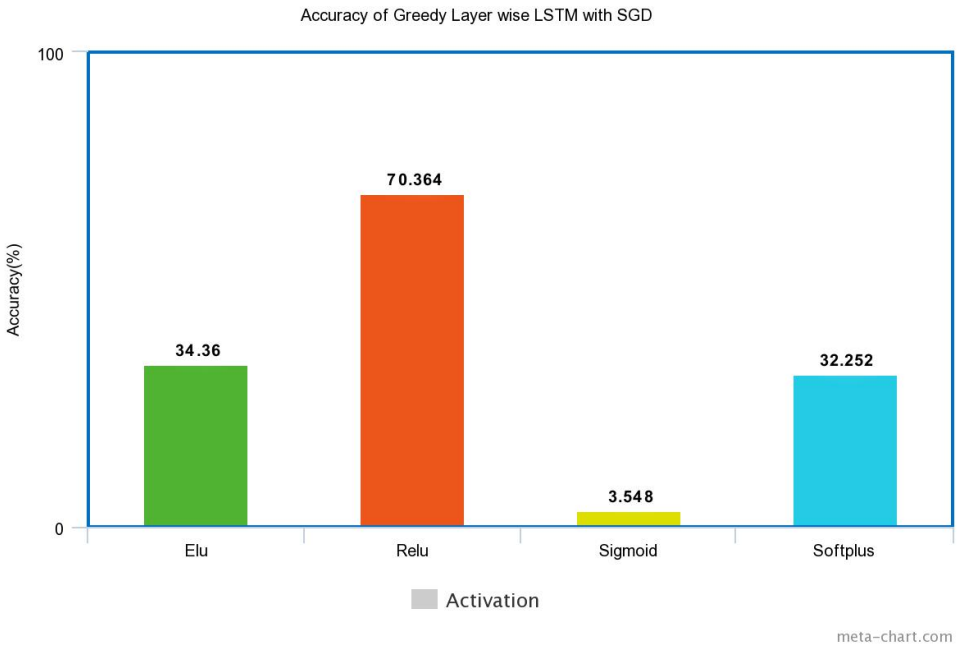

Figure 24: Accuracy of Stochastic Gradient Descent Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	0	14696	52646	0
DoS	0	0	1052	44875	0
Probe	0	0	3533	8123	0
U2r	0	0	12	983	0
R2L	0	0	0	52	0

Table 3.84: Confusion matrix for Sigmoid Activation and SGD optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.00	0.00	0.00	45927
Probe	0.18	0.30	0.23	11656
U2R	0.01	0.99	0.02	995
R2L	0.00	0.00	0.00	52
avg/total	0.02	0.04	0.02	125972

Table 3.85: Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	36009	907	5675	13761	10990
DoS	929	41970	1235	1776	17
Probe	20	362	10253	705	316
U2r	1	52	9	368	565
R2L	4	2	0	7	39

Table 3.86: Confusion matrix for reLU Activation and SGD optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.97	0.53	0.69	67342
Dos	0.97	0.91	0.94	45927
Probe	0.60	0.88	0.71	11656
U2R	0.02	0.37	0.04	995
R2L	0.00	0.75	0.01	52
avg/total	0.93	0.70	0.78	125972

Table 3.87: Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	2080	1066	34496	29700
DoS	0	40175	787	654	4311
Probe	0	3164	2503	4069	1920
U2r	0	5	1	586	403
R2L	0	0	1	21	30

Table 3.88: Confusion matrix for eLU Activation and SGD optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.88	0.87	0.88	45927
Probe	0.57	0.21	0.31	11656
U2R	0.01	0.59	0.03	995
R2L	0.00	0.58	0.00	52
avg/total	0.38	0.34	0.35	125972

Table 3.89: Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	2326	24	11513	53479
DoS	1	40176	537	816	4397
Probe	0	5246	271	364	5775
U2r	0	5	0	134	856
R2L	0	20	0	4	48

Table 3.90: Confusion matrix for Softplus Activation and SGD optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.84	0.87	0.86	45927
Probe	0.33	0.02	0.04	11656
U2R	0.01	0.13	0.02	995
R2L	0.00	0.92	0.00	52
avg/total	0.34	0.32	0.32	125972

Table 3.91: Precision, Recall, and F1 Score of LSTM with Stochastic Gradient Descent Optimizer for Softplus Activation

3.5.5 Adadelta Optimizer

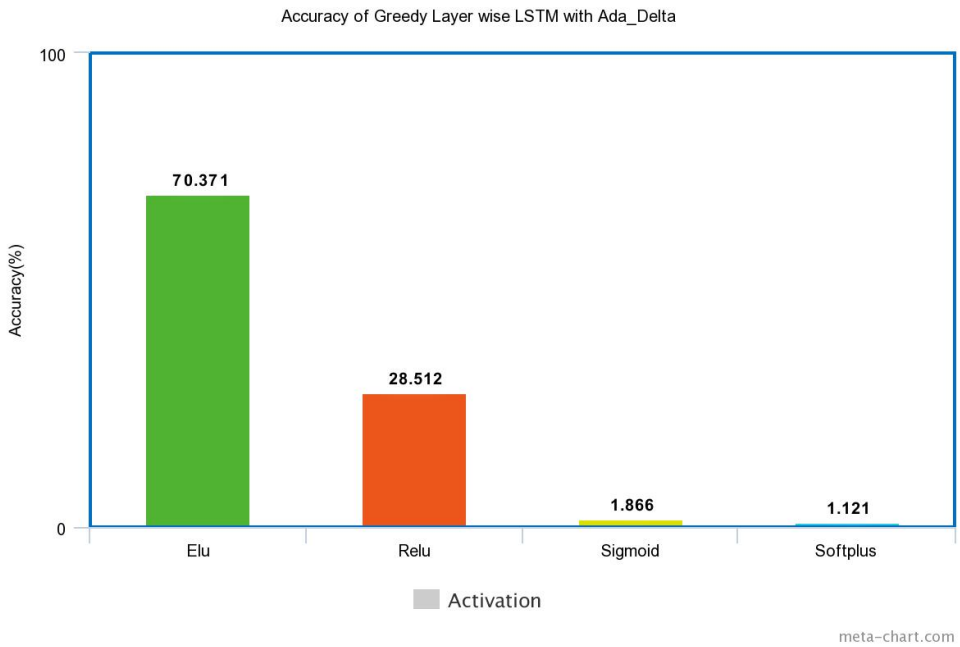

Figure 25: Accuracy of Adadelta Optimizer for different activation functions

Sigmoid Activation

	Normal	Dos	Probe	U2r	R2L
Normal	2299	0	0	0	65043
DoS	40164	0	0	0	5763
Probe	5138	0	0	0	6518
U2r	3	0	0	0	992
R2L	0	0	0	0	52

Table 3.92: Confusion matrix for Sigmoid Activation and Adadelta for LSTM

	Precision	Recall	F1-score	Support
Normal	0.05	0.03	0.04	67342
Dos	0.00	0.00	0.00	45927
Probe	0.00	0.00	0.00	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	1.00	0.00	52
avg/total	0.03	0.02	0.02	125972

Table 3.93: Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for Sigmoid Activation

ReLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	16715	774	722	49131
DoS	0	34663	6891	701	3672
Probe	0	5954	1226	2546	1930
U2r	0	542	0	1	452
R2L	0	24	0	0	28

Table 3.94: Confusion matrix for reLU Activation and Adadelta optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.60	0.75	0.67	45927
Probe	0.14	0.11	0.12	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	0.54	0.00	52
avg/total	0.23	0.29	0.25	125972

Table 3.95: Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for ReLU Activation

eLU Activation

	Normal	Dos	Probe	U2r	R2L
Normal	36026	907	5639	13763	11007
DoS	929	41970	1235	1776	17
Probe	20	360	10245	715	316
U2r	2	52	8	368	565
R2L	4	2	0	7	39

Table 3.96: Confusion matrix for eLU Activation and Adadelta optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.97	0.53	0.69	67342
Dos	0.97	0.91	0.94	45927
Probe	0.60	0.88	0.71	11656
U2R	0.02	0.37	0.04	995
R2L	0.00	0.75	0.01	52
avg/total	0.93	0.70	0.78	125972

Table 3.97: Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for eLU Activation

Softplus Activation

	Normal	Dos	Probe	U2r	R2L
Normal	0	0	989	0	66353
DoS	0	0	6663	0	39264
Probe	0	0	1361	0	10295
U2r	0	0	0	0	995
R2L	0	0	0	0	52

Table 3.98: Confusion matrix for Softplus Activation and Adadelta optimizer for LSTM

	Precision	Recall	F1-score	Support
Normal	0.00	0.00	0.00	67342
Dos	0.00	0.00	0.00	45927
Probe	0.15	0.12	0.13	11656
U2R	0.00	0.00	0.00	995
R2L	0.00	1.00	0.00	52
avg/total	0.01	0.01	0.01	125972

Table 3.99: Precision, Recall, and F1 Score of LSTM with Adadelta Optimizer for Softplus Activation

Chapter 4

Conclusion

From this project we can conclude that testing a model for different activation function and for different optimization algorithms yields different results on same data.

The variation in accuracies arises from the nature of the activation functions and the optimization algorithms. As it is seen from the above results Convolutional Neural Network yields better result than Autoencoder and LSTM with Tanh as activation function for adam optimizers. We mentioned CNN as better due to the accuracy that it shows compared to all other accuracies considering all the parameters for other neural networks that we implemented

Considering the time complexity , Autoencoder performs better compared to CNN and LSTM.we have considered 10 epochs as a criteria for justifying time complexity.Result stand out that LSTM takes on an average 10 minutes for one epochs, and Autoencoder will complete all 10 epochs in just 5-6 minutes, on the other hand a CNN takes around 8-9 minutes for 10 epochs. Time complexity also depends on how the network structure is. On an average if we have similar structure then also LSTM takes more time compared to other neural networks

The data we used was clean and preprocessed free of an ambiguity but in real life scenario the data will be corrupt with a lot of ambiguities for which the network has to be robust. Given the performance of all the models above , we can decide which model we want to implement.

Bibliography

[1] Threat Landscape for Industrial Automation System in Second Half of 2016, Kaspersky Lab ICS CERT.

[2] INTRUSION DETECTION SYSTEMS;DEFINITION, NEED AND CHALLENGES, SANS Institute 2001,

[3] Anderson, Ross (2001). Security Engineering: A Guide to Building Dependable Distributed Systems. New York: John Wiley & Sons. pp. 387–388. ISBN 978-0-471-38922-4.

[4] Limitations of Network Intrusion Detection, Steve Schupp, December 1, 2000, https://www.giac.org/paper/gsec/235/limitations-network-intrusion-detection/100739

[5] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature. 405. pp. 947–951.

[6] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter, latest version 22 Feb 2016 (v5).Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289 [cs.LG]

[7] Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). ”Generative Adversarial Networks”. arXiv:1406.2661

[8] On the momentum term in gradient descent learning algorithms, NingQian, Qinghua Hu, Yun Wang, Zongxia Xie, Pengfei Zhu, Daren Yu Center for Neurobiology and Behavior, Columbia University, 722 W. 168th Street, New York, NY 10032, USA

[9] An overview of gradient descent optimization algorithms, Sebastian Ruder, Cited from: arXiv:1609.04747v2

[10] Large-Scale Machine Learning on Heterogeneous Distributed Systems, Preliminary White Paper, November 9, 2015

[11] Jack Clark. Google turning its lucrative web search over to AI machines, 2015,www.bloomberg.com/news/articles/2015-10-26/googleturning-its-lucrative-web-search-over-to-ai-machines.

[12] Improving DNNs for LVCSR using rectified linear units and dropout, George E. Dahl, Tara N. Sainath, Geoffrey E. Hinton, 2013

[13] A Practical Guide to Training Restricted Boltzmann Machines, I. Aleksander, M. De Gregorio, F.M.G. França, P.M.V. Lima, H. Morton,January 2009

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In Parallel Distributed Processing. Vol 1: Foundations. MIT Press, Cambridge, MA, 1986.

[15] Autoencoders, Unsupervised Learning, and Deep Architectures, Pierre Baldi, Department of Computer Science,University of California, Irvine, 2010

[16] Alireza Makhzani, Brendan Frey, k-Sparse Autoencoders, 19 Dec 2013, arXiv:1312.5663 [cs.LG]

[17] Hinton, G. (2009). "Deep belief networks". Scholarpedia. 4 (5): 5947. doi:10.4249/scholarpedia.5947.

[18] Smolensky, Paul (1986). "Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory" (PDF). In Rumelhart, David E.; McLelland, James L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press. pp. 194–281. ISBN 0-262-68053-X.

[19] Ruslan Salakhutdinov and Geoffrey Hinton (2010). Replicated softmax: an undirected topic model. Neural Information Processing Systems.

[20] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Greedy Layer-Wise Training of Deep Networks, Dec 2006, University of Montreal.

[21] Matthew D. Zeiler. ADADELTA: AN ADAPTIVE LEARNING RATE METHOD,Google Inc., USA, New York University, USA.