Network Intrusion Detection

A PROJECT REPORT

Submitted in partial fulfillment of the

Requirement for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & CLOUD COMPUTING ENGINEERING

Submitted by

Vaibhav Dingreja (2011775)

Under the supervision of

Ms. Komal Singh
(Associate Professor of GEU)

DEPARTMENT OF COMPUTER SCIENCE & CLOUD COMPUTING ENGINEERING

GRAPHIC ERA DEEMED TO BE UNIVERSITY, DEHRADUN – 248002 (INDIA)

JUNE, 2020

Network Intrusion Detection

A PROJECT REPORT

Submitted in partial fulfillment of the

Requirement for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & CLOUD COMPUTING ENGINEERING

Submitted by

Vaibhav Dingreja (2011775)

Under the supervision of

Ms. Komal Singh
(Associate Professor of GEU)

DEPARTMENT OF COMPUTER SCIENCE & CLOUD COMPUTING ENGINEERING

GRAPHIC ERA DEEMED TO BE UNIVERSITY, DEHRADUN – 248002 (INDIA)

JUNE, 2020

GRAPHIC ERA DEEMED TO BE UNIVERSITY, DEHRADUN

CANDIDATE DECLARATION

I hereby declare that the project titled "**Network Intrusion Detection**", which is being submitted by me in the partial fulfillment of the requirement for the award of Degree of **Bachelor of Technology** in **Computer Science and Cloud Computing Engineering**, submitted to the **Department of Computer Science Engineering, Graphic Era (Deemed To Be) University Dehradun,** is an authentic record of my own work carried out during the period of July 2019 to June 2020 under the guidance of Ms. Komal Singh, Associate Professor, Department of Computer Science Engineering, Graphic Era Deemed to be University, Dehradun.

I have not submitted the matter embodied in this project for the any award of any other degree.

Date: - 14/6/2020

Dehradun **Vaibhav Dingreja**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Ms. Komal Singh
Associate Professor
Department of Computer Science,
Graphic Era Deemed to be University,
Dehradun.

Dr. Devesh Pratap Singh
Head of Department
Department of Computer Science,
Graphic Era Deemed to be University,
Dehradun.

Examiner

ACKNOWLEDGEMENT

I would like to take the opportunity to thank all those people who constantly motivated and provided me with inspirational guidance during working of this project.

Before I introduce my final year project on "**Network Intrusion Detection**", I would like to add few heartfelt words for the people who are part of this project. I wish to thank **Dr. Kamal Ghanshala Hon'ble Chairman, Graphic Era (Deemed to Be) University, Dehradun** for providing us the dedicated faculty and well equipped labs which played the great role in completion of this project.

I would like to thank my project guide **Ms. Komal Singh** for her constant motivation. Without her valuable guidance, it would not have been possible for me to complete this project. She was always there for me in any obstacle.

I would also like to thank the Head of Department Computer Science **Dr. Devesh Pratap Singh** for his unconditional support. A special thanks to the lab staff and faculty members who have imparted us the knowledge which played a vital role in the completion of the project.

I would like to conclude while expressing my deepest gratitude to my parents who were always my strongest pillars of wisdom.

Date: 14/6/2020

Place: Dehradun **Vaibhav Dingreja (2011775)**

ABSTRACT

To secure a network from intrusion and for the confidentiality of any data, an Intrusion Detection System plays a vital role. The main objective is to achieve an accurate performance of an NIDS system which adepts in detection of various types of attacks in the network. In this report, we have explored the performance of a Network Intrusion Detection System (NIDS) which can detect attacks in the network using Deep Reinforcement Learning Algorithm (DRL). We have exploited Deep Q Network algorithm which is a value-based Reinforcement Learning algorithm technique used in detection of network intrusions. Moreover, we have analyzed the accuracy of our model in comparison with different types of models. In this paper, we illustrated the comparison of our NIDSDQN model to a previous model designed in other approaches like J48, artificial neural network, random forest, support vector machine. Our Goal is to detect attacks without depending on the past experience and at its first attempt. We used data set for minimizing the false alarm rate. Previous work was based on a benchmark dataset such as KDD-99, NSL-KDD, which shares the same attributes for all models. We have worked on similar type of dataset which aided as an effective means in detection of different types of attacks. The Deep Q Network-Intrusion Detection System (DQN-IDS) model improves the accuracy and performance an IDS and provides a new means as a research method for intrusion detection.

Keywords: Network Security, Machine Learning, Deep Q Network, Artificial Neural Network, DDoS, NIDS

TABLE OF CONTENTS

Content **Page No.**

CHAPTER 4: Software Design

CHAPTER 5: Hardware and Software Requirements

CHAPTER 6: Coding/Code Templates

CHAPTER 7: Testing

CHAPTER 8: Output Screens

CHAPTER 9: Conclusion……52

CHAPTER 10: Further Enhancements and Recommendations…….53

LIST OF FIGURES

LIST OF ABBREVATIONS

S.No	Abbreviations	Full Form
1.	NIDS	Network Intrusion Detection System
2.	ANN	Artificial neural network
3.	CNN	Convolutional Neural Network
4.	KNN	K Nearest Neighbor
5.	GRU	Gated Recurrent Unit
6.	LSTM	Long Short Term Memory
7.	DNN	Deep Neural Network

CHAPTER 1
INTRODUCTION

1.1 Overview

The accurate and timely identification of an intruder has become of paramount importance over the past decade as the advent of the Internet has brought with it a new breed of criminals. The importance of protecting your computer system from hackers has become greatly increased as communication channels that are available both for the distribution of malicious software, and expertise in exploitation of computer systems have open up. Network Security is a broad term that covers a multitude of technologies, devices and processes. In its simplest term, Network security is the practice of preventing and protecting against unauthorized intrusion into corporate networks. It is the process of taking physical and software preventive measures to protect the underlying network infrastructure from unauthorized access, misuse, malfunction, modification, destruction, or improper disclosure, thereby creating a secure platform for computers, users and programs to perform their permitted critical functions within a secure environment. To implement this kind of defence in depth, there are a variety of specialized techniques and types of network security. Today's network architecture is complex and is faced with a threat environment that is always changing and attackers that are always trying to find and exploit vulnerabilities. These vulnerabilities can exist in a broad number of areas, including devices, data, applications, users and locations. For this reason there are many security management tools and applications in use today that address individual threats and exploits and also regulatory non-compliance. When just a few minutes of down-time can cause wide disruption and massive damage to an organization's bottom line and reputation, it is essential that these protection measures are in place. An intrusion detection system (IDS) is a device or software application that monitors a network for malicious activity or violation is typically reported or collected centrally using a security information and event management system. When placed at a strategic point or points within a network to monitor traffic to and from all devices of the network, an IDS will perform an analysis of passing traffic, and match the traffic that is passed on the subnets to the library of known attacks. Once an attacked is identified, or abnormal behaviour is sensed, the alert can be sent to the administrator. There are many layers to consider when addressing network security across an organization. Attacks can happen at any layer in the network security layer model, so your network security hardware, software and policies must be designed to address each area. Modern networked business environments require a high level of security to ensure safe and trusted communication of information between various organizations. An intrusion detection system can act as an adaptable safeguard technology for system security after traditional technologies fail. Cyber-attacks will only become more sophisticated, so it is important that protection technologies adapt along with their threats. IDS and firewall both are related to the network security but an IDS differs from a firewall as firewall looks outwardly for intrusions in order to stop them from happening. Firewalls restrict access between networks to prevent intrusion and if attack is from inside the network it doesn't signal. An IDS describes a suspected intrusion once it has happened and then signals an alarm.

In addition to protecting assets and integrity of data from external exploits, network security can also manage network traffic more efficiently, enhance network performance and ensure secure data sharing between employees and data sources. Machine learning uses algorithms for parsing data, learn from that parsed data, and make informed and intelligent decisions based on what it has learned. Artificial Neural Network has created using deep learning structures in layers which can learn and make decision on its own. Our approach proposes Deep Q learning based approach which can provide a new insight to overcome the difficulties and challenges of developing and reliable and efficient Network Intrusion Detection System. We can collect network traffic data from network sources after a good attribute representation of the datasets using deep Q learning approach. The data of labelled network traffic can be collected from a private and isolated environment. We have used deep Q learning approach to develop a NIDS to detect any known or unknown type of attack.

1.2 Problem Statement

Network Intrusion Detection System to detect types of attacks and training machine using Wireshark, Feature Extraction and Machine Learning Algorithms.

1.3 Motivation

As we all know that digitization has transformed our world. How we live, work, play and learn have all changed. Every organization that wants to deliver the services that customers and employees demand must protect its network. Network security also helps you protect proprietary information from attack. Before deploying an intrusion detection system, organisations should consider commissioning an independent risk assessment to better understand their environment, including the key assets requiring protection. Being armed with this knowledge will help to ensure that an IDS is properly scoped to ensure that it offers the greatest value and benefits. Given the challenges of ongoing system maintenance, monitoring and alert investigation, many organisations may wish to consider enlisting a managed service to perform all the heavy lifting. A managed IDS service avoids the need to recruit dedicated security personnel, and if necessary, can also include all requisite technology, circumventing the need for upfront capital expenditure. To attain a high level of threat visibility, organisations must ensure that intrusion detection technology is correctly installed and optimised. ID requires significant expertise, both technical and organizational. Professionals need to have knowledge of their own unique network environment, since what is classified as a security event in one network may not be considered one in another network.

The aspiration of this project is to give an evaluation of the performance and efficiency of algorithms which we have used that will redirect any other who wishes to use the

approaches to know and learn about the accuracy under certain condition in an organizational network. Moreover, a novel and new evaluation technique has been considered. We can consider and evaluate this approach by their speed and accuracy. We also consider examine the behaviour and result of using new dataset on this algorithm.

1.4 Challenges

While deploying a flexible and efficient NIDS for unknown future attacks we have faced two challenges. Firstly, proper attribute selection of network traffic dataset is difficult for unknown attacks. The attribute selected for one kind of attack may not be compatible for detecting another kind of attack. When this happens, NIDS treats that traffic as normal or show some error. Second challenge is unavailability of labelled real time traffic dataset to develop a NIDS system. It would need an immeasurable effort to label a real time traffic, a raw data with respect to different attacks. Available datasets which are labelled are quite old and contains less attributes to detect any new or unknown attack**.**

1.5 Working of Network Intrusion Detection System

The Intrusion Detection System detects the point of attack in the system using various machine learning model and detects the IP address that is sending attack. It is trained on the dataset of previous attacks using Network Capture tool and feature extraction tool to extract more features and train our models in best possible way. It is a comparison based model where it gives the accuracy and precision of detecting the attacks using normal machine learning models like SVM, KNN, Random Forest , Decision Tree, GaussianNB, ADABoost Classifier, Logistic Regression and Grid Search Algorithm.

Then it compares with the new mixed model of Convolutional Neural Network, Recurrent Neural Network, Long Short-Term Memory Model, Gated Recurrent Unit and Deep Neural Network for best precision. Once we compared the result it is then we made a new model using the concepts of Reinforcement Learning with Deep Q learning where we give rewards to our agent for predicting the right result and hence making a better model than above two models with higher accuracy. Our Model also detects if the traffic coming is the normal traffic or the attack and hence plotting a confusion matrix and a true prediction graph on it.

1.6 Detection Method Of IDS

Signature-Based Method: Signature based IDS detects the attack on the basis of the specific patterns such as number of bytes or number of 1’s or number of 0’s in the network traffic. It also detects on the basis of the already known malicious intrusion sequence that is used by malware. The detected patterns in the IDS are known as

Signatures. Signature-based IDS can easily detect the attacks whose pattern (signature) already exists in the system but it is quite difficult to detect the new malware attacks as their pattern is now known.
Anomaly-based Method: Anomaly based IDS was introduced to detect the unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in a model. Machine Learning based method has a better generalized property in comparison to signature-based IDS as these models can be trained according to the hardware and application configuration.

1.7 Applications

Network Intrusion Detection System has a wide variety of applications. Some of them are listed below:

1. Banking

Banks maintain very specific hours so security operations must be very specific as well. An intelligent bank solution will have a custom plan for after hours, open hours and while the bank is opening and closing. After hours intrusion detection at base line is very similar to other facilities. Motion detectors, Glass Break detectors, Seismic Activity detectors and door contacts are primary line of defence. These devices are connected, wired or wirelessly to an intrusion detection alarm panel within the bank. The intrusion detection system can be programmed with dual authentication system.
In Banking System to secure the transaction history of an individual person to secure the network of online banking and transactions we need an IDS which can immediately detect any upcoming attack and can save the economy of the company. We have seen many cases in history where the network of online transaction was attacked and the intruder causes the loss of large money for an organization or person.

2. Cloud Computing

Cloud Computing is rapidly growing computational model in today's IT world. It delivers convenient, on-demand network access to shared pool of configurable computing resources. The distributed and open structure of cloud computing and services becomes an attractive target for potential cyber-attacks by intruders. Hence it is vulnerable and prone to intrusions that affect confidentiality, availability and integrity of cloud resources and offered services.

3. Military and Government Sites

Military Communication Network and Government sites are one of the most important sites which needs to be protected from terrorists and other countries intruders. Hence the use of Intruder Detector System in such sites even increases more and the need for

accuracy and precision for detecting an attack must be one hundred percent in these sites, as they contain a lot of sensitive data regarding the country and its security. As breach in any of the sites may cause loss of lives and economy for a country. Hence, we apply the best IDS in network traffic passing points for such sites.

4. Organizations

Due to extensive usages of internet, electronic assaults on network and information system of the financial organizations, military and energy sectors are increasing. Large web sites of any organization are attacked by various intruders and hackers. The information security is very important aspect to protect the valuable data of any organization. The organizations need security applications that effectively protect their networks from malicious attacks and misuse. Intrusion detection system can detect the intrusions and protect the information system from the security violations. It is used to identify the malicious use of computer and computer network. It detects access to un-authorized user, the violation of security and finds illegal users. IDS ensure application availability with zero-day protection and industry-leading security research.

5. Aviation Sectors

The need to protect ATC systems from cyber-attacks requires enhanced attention because the Federal Aviation Administration (FAA) has increasingly turned toward the use of commercial software and Internet Protocol (IP) 1-based technologies to modernize ATC system. Now, attackers can take advantage of software vulnerabilities in commercial IP products to exploit ATC systems, which is especially worrisome at a time when the Nation is facing increased threats from sophisticated nation-state-sponsored cyber-attacks. To effectively monitor and detect potential cyber-security incidents on a network, intrusion-detection-system (IDS) sensors need to be installed at various critical network points. There, sensors automatically generate security alerts when potential cyber-attacks are detected so that further incident response can be made. FAA's intrusion-detection capability is ineffective because of inadequate deployment of IDS sensors at the facility level and a lack of timely remediation of incidents detected. Specifically, ATC systems are located at hundreds of operational facilities such as en route centers, terminal radar approach control facilities, and airport control towers. However, IDS sensors have been deployed to only 11 of these ATC facilities. Further, none of the IDS sensors are installed to monitor ATC operational systems at these facilities, such as the IP-based network associated with the Host Computer System. Instead, these sensors provide monitoring coverage only for mission-support systems, such as email systems. IDS system ensures that the ATC system is protected from both physical and cyber security threats to prevent disruptions in air travel and commerce.

CHAPTER 2
LITERATURE SURVEY

Several Techniques are available in the literature for detecting the intrusion behaviour. In recent times, intrusion detection has received a lot of interest among the researchers since it is widely applied for preserving the security within a network. Previously a lot of work has been done in to detect the network intrusion such as anomaly based or misuse detection.

A.M. Chandrasekhar, K. Raghuveer et al. [1] proposed a technique which is divided into four steps: initial step, k-means clustering is used to generate different training subset then based on the obtained subset, various neuro-fuzzy data model are trained. Consequently, a vector for SVM classification is obtained and in last, classification using radial SVM is applied to detect the intrusion occurred or not. To demonstrate the applicability and ability of the new method, the result of KDD dataset is confirmed in which it shows that the proposed methods produce better result than the BP, multiclass SVM and other approach such as decision tree.

Sumaiya Thaseen, Ch. Aswani Kumar et al. [2] proposed dissimilar tree based categorization algorithms that categorize network events in intrusion detection systems and the experiment is performed using NSL-KDD 99 dataset. Dimensionality of the element of the dataset is reduced. The outcomes show that Random Tree model embraces the highest degree of correctness and reduced false alarm rate. This model is calculated with other leading intrusion detection models to conclude its better predictive accuracy.

Nidhi Srivastav, Rama Krishna Challa et al. [3] present the layered framework integrated with neural network to construct an effective and efficient intrusion detection system. Such system has experimented with Knowledge Discovery & Data Mining (KDD) 1999 dataset. This system was compared with presented approaches of intrusion detection which either uses neural network or based on layered framework. The results illustrate that the proposed method has high attack detection accuracy and less false alarm rate.

Zhao Y ongli, Zhang Yungui, Tong Weiming, Chen Hongzhi et al. [4] proposed the method which is based on MAHALANOBIS Distance characteristic ranking and an improved comprehensive search to choose an improved combination of features. They evaluated the approach on the KDD CUP 1999 datasets using SVM classifier and KNN classifier. The results showed that classification is done with high classification rate and low misclassification rate with the reduced feature subsets.

Fengli Zhang, Dan Wang et al. [5] proposed an effective feature selection approach based on Bayesian Network classifier and with the identical intrusion detection benchmark dataset (NSL-KDD), the performance of the proposed approach is calculated and compared with other usually used feature selection methods which is shown by empirical results that features selected by this approach have decreased the

time to detect attacks and increased the classification precision as well as the true positive rates significantly.

Yang Li et al. [6] proposed a lightweight intrusion detection method that was comprehensively efficient and successful based on feature selection. In this both IG and ChiSquare approach is combined in order to reduce the possibility that a single feature selection approach would result to some biased results. However, in IG method, the bias in favour of multi-valued features is a well-known problem and in ChiSquare method, there is a major limitation that it assumes the independence property of features. Since the interaction of appropriate features is not taken into account, sorting on highly correlated features may disgrace the performance of classifiers.

Preecha Somwang, Woraphon Lilakiatsakun et al. [7] proposed the clustering technique by using hybrid method based on Principal Component Analysis (PCA) and Fuzzy Adaptive Resonance Theory (FART) for identifying various attacks. The PCA is apprehensive to random selects the best provenance and reduction the feature space. The FART is implementing which is used to classifying variation collection of data, regular and irregular. The proposed method can advances the high performance of the detection rate and to reduce the false alarm rate and this is computed approach on the benchmark data from KDD Cup 99 data set.

S. F. Owens and R. R. Levary [8] have stated that intruder detection systems have been commonly constructed using expert system technology. But, Intrusion Detection System (IDS) researchers have been biased in constructing systems that are difficult to handle, lack insightful user interfaces and are inconvenient to use in real-life circumstances. The proposed adaptive expert system has utilized fuzzy sets to find out attacks. The expert system comparatively easy to implement when used with computer system networks has the capability of adjusting to the nature and/or degree of the threat. Experiments with Clips 6.10 have been used to prove the adjusting capability of the system. Alok Sharma have focused on the use of text processing techniques on the system call sequences for intrusion detection. Host-based intrusions have been detected by introducing a kernel based similarity measure. Processes have been classified either as normal or abnormal using the k-nearest neighbor (kNN) classifier. They have assessed the proposed method on the DARPA-1998 database and compared its operation with other existing methods present in the literature.

Shi-Jinn Horng [9] have used a combination of hierarchical clustering algorithm, easy feature selection method, and SVM technique in their proposed SVM-based intrusion detection system. Fewer, abstracted, and higher-qualified training instances that are derived from the KDD Cup 1999 training set has been given to the SVM by the hierarchical clustering algorithm. The simple feature selection method employed for the removal of insignificant features from the training set has enabled the proposed SVM

model to achieve more precise classification of the network traffic data. The proposed system has been assessed using the renowned KDD Cup 1999 dataset. Compared to other intrusion detection systems that are based on the same dataset, the proposed method has exhibited superior performance in identifying DoS and Probe attacks and an overall best performance in accuracy.

B. Shanmugam and Norbik Bashah [10] Idris have proposed an advanced fuzzy and data mining methods based hybrid model to find out both misuse and anomaly attacks. Their objective was to decrease the quantity of data kept for processing and also to improve the detection rate of the existing IDS using attribute selection process and data mining technique respectively. A modified version of APRIORI algorithm which is an improved Kuok fuzzy data mining algorithm utilized for implementing fuzzy rules has enabled the generation of if-then rules that show common ways of expressing security attacks. They have achieved faster decision making using mamdani inference mechanism with three variable inputs in the fuzzy inference engine which they have employed. The DARPA 1999 data set has been used to test and benchmark the efficiency of the proposed model. In addition, the test results against the "live" networking environment within the campus have been analysed.

O. A. Adebayo [11] has presented a method that uses Fuzzy-Bayesian to detect real-time network anomaly attack for discovering malicious activity against computer network. They have established the effectiveness of the method by describing the framework. The overall performance of the intrusion detection system (IDS) based on Bayes has been improved by a combination of fuzzy with Bayesian classifier. In addition, by the experiment carried out on KDD 1999 IDS data set, the practicability of the method has been verified.

Abadeh, M.S. and Habibi, J. [12] have proposed a method to develop fuzzy classification rules for intrusion detection use in computer networks. The method of fuzzy rule base system design has been based on the iterative rule learning approach (IRL). Using the evolutionary algorithm to optimize one fuzzy classifier rule at a time, the fuzzy rule base has been created in an incremental fashion. Intrusion detection problem has been used as a high-dimensional classification problem to analyse the functioning of the final fuzzy classification system. Results have demonstrated that the fuzzy rules generated by the proposed algorithm can be utilized to build a reliable intrusion detection system.

Arman Tajbakhsh [13] have presented a data mining technique-based framework for constructing an IDS. In the framework, Association Based Classification (ABC) has been used by the classification engine which is in fact the central part of the IDS. Fuzzy association rules have been used by the proposed classification to construct classifiers. Some matching measures have been used to evaluate the consistency of any new sample (which is to be categorized) with various class rule sets and the label of the sample has been declared as the class that is analogous to the best matched rule set. A method which decreases the items that may be included in extracted rules has also been proposed to reduce the time taken by the rule induction algorithm. The framework has been assessed using KDD-99 dataset. The results have shown that the achieved total detection rate and detection rate of known attacks are large and false positive rate is small, though the results are not bright for unknown attacks.

CHAPTER 3
SOFTWARE REQUIREMENT ANALYSIS

3.1 Introduction

We are developing new model of Intrusion Detection System which has capacity of self-detecting or updating attacks. In proposed IDS model we have developed Artificial Neural Network algorithm with Reinforcement Deep Q learning algorithm to detect attacks and accuracy of it. In proposed model we define two separate sets of data. 1] Training set 2] Testing set. We used KDD Cup dataset as standard dataset for network trafficking and extract the features in our own dataset using the feature extraction tool like floatibag and hence making 32 features of it to train our model more accurately.

3.2 Document purpose

The Project concept is to achieve the new method for extracting the information from the Dataset that we have collected using Wireshark network capturing tool that will help to automatically detect the attack like Dos. How such attack are Detected by Machine Learning and ANN module and provide the security from the any anomaly data which attacks your system and provide security to the server.

3.3 Intended audience and reading suggestions

This document is intended to be read by the customers like net developers, project managers, staff, users, testers and documentation writers. This is a technical document and the terms should be understood by the customers. This document is intended for: Developers: In order to be sure they are developing the right project that fulfil requirements that provided in this document. Testers: In order to have an exact list of the features and functions that has to respond according to requirements and provided diagrams. Users: In order to get familiar with the idea of the project and suggest other features that would make it even more functional. Documentation Writers: To know what features and in what way they have to explain. What security technologies are required, how the system will response on each users action etc. Advanced end users, end users/desktop and system administrators: In order to know exactly what they have to expect from the system, right inputs and outputs and response in error situations.

3.4 Product scope

One of the most important issues about our proposed architecture is detecting the attack using intrusion detection system, in order to verify predictions of the system, to determine the requirements of the intrusion detection system and to evaluate the use of neural network and reinforcement learning within the problem domain.

3.5 Overall Description

3.5.1 Product perspective

This feature will give the user a system which can detect the upcoming attack and which type of attack it is. Hence, giving the user a more secure system to detect the attack and learning the machine using reinforcement learning model. With reinforcement learning model we are not training our machine with one specific model but our machine will train itself even on new database records and propose a model by itself which is best for detecting.

3.5.2 Product functions

In this extraction framework, intermediate output of IDS is stored so that only the improved component has to be deployed to the entire database. Extraction is then performed on both the previously processed data from the unchanged components as well as the updated data generated by the improved component by different ANN and compare it with Reinforcement Deep Q-Learning Algorithm. Performing such kind of incremental extraction can result in a tremendous reduction of processing time. The project provides the following functions:

1. Determine the requirements of an intrusion detection system
2. Evaluate the use of artificial neural networks within the problem domain.
3. Determine the most appropriate type of artificial neural network to use
4. Develop an intrusion detection system using the techniques that are found to be best suited to the problem.
5. Train and test the intrusion detection system using test data.
6. Test the system on unseen data.

3.6 System Features

3.6.1 System Feature 1

Functional Requirements:

1.The user will be able collect the necessary training data using an appropriate tool.

2. The user will be able to train the neural network based analysis engine.

3. The system must be able to be started and stopped by a privileged user of the system.

4. The system is required to notify a system administrator when a sequence of events that is likely to be an attack is encountered.

5. Statistics of the systems operation must be able to be collected for analysis and review.

3.6.2 System Feature 2

Quality and Efficiency: Using the training and testing set automatically detect the attacks. Our approach minimizes the need of reprocessing the entire collection of attack in the presence of new extraction goals and deployment of improved processing speed of the server.

3.7 Other Non-functional Requirements

3.7.1 Performance Requirements

The system must be easy to train for new users of the software. The mechanism for training the system used must be easy to understand and allow someone with a reasonable background in computer networks to capture the correct data and use the training software to create an analysis engine for their network. Users of the system must be able to start and stop the system easily. Users of the system must be able to configure any parameters the system may use in an easily manageable way.

3.7.2 Software quality attributes

1.Adaptability: The compiler must be able to accommodate changes to the language implementation as well changes in the machine architecture.

2.Correctness: The compiler generated code should give the exact output as that of the output of the script run using the interpreter.

False negative identifications of attacks should be minimized heavily. This means a higher focus on detecting false negatives should be given to the project rather than trying to reduce false positives, although a minimized false positive rate is desirable as a secondary concern.

3.7.3 Performance

Packets must be captured fast enough to enable real-time analysis of the data, and a system administrator should be notified as soon as possible. When the service becomes slow due to heavy traffic, the system should not affect the performance of the rest of network. The system should express a high degree of tolerance to internal faults and external faults with the system itself as well as its host system.

3.7.4 Supportability

Dynamic configuration of the system will allow the network administrators to add a different weight matrix to the system with minimal disruption to the service, and without disrupting any other services on the network.

3.8 Problem Statement

Our main aim is to provide an additional approach for identifying/ stopping an attack with the use of Reinforcement Learning techniques. The objective of work is to

analyse various types of attacks on a network system and improve the detection accuracy rate with new type of dataset by machine learning in network security domain.

Our main research questions are as follows:

• Problemstatement1.1: How to enhance the network intrusion detection system with the use of Reinforcement Learning algorithm by obtaining the optimal policy with the maximum reward?

• Problemstatement1.2: What type of dataset are we going to use while training the model? How the data set will affect the performance of Machine Learning Algorithms?

3.9 Technical Model

A technical model is designed to provide an overview of all the functionalities and the features. The Reinforcement algorithms will be developed and combined to work together with the NIDS system in order to obtain the optimal policy and detect the network intrusions and to save the system resources. The technical model can be described as the following steps

• System monitoring

• Decision with changes in the NIDS system like reordering or updating the variables/attributes.

It is necessary to find the cause of network intrusion thus, it is important to monitor the system resources parameter, such as CPU and RAM to find out which resource metric that could be the bottleneck that makes packets to be dropped or intruded which may be an active attacks which cause interruption, modification, or Fabrication. And among passive attacks like Release of message content or Traffic Analysis. The most important prerequisites that must be in order before conducting any implementation of the Deep Q Learning algorithm are finding the right system resource parameter to monitoring part and deployment of the underlying infrastructure.

3.9.1 Domain

The best advantage for a system is when an IDS serves localhost and users over the internet, which is installed on the server side as shown in figure 3.1. There are four main important things in the system which are the system administrator, monitor, user, and network. The User sends a request to the server over the internet or Local Area Network and the Intrusion Detection System will analyse the packets which are received by the server. This Intrusion Detection System can detect intrusions both internally and externally. It alerts system administrator as soon as it detects an intrusion.

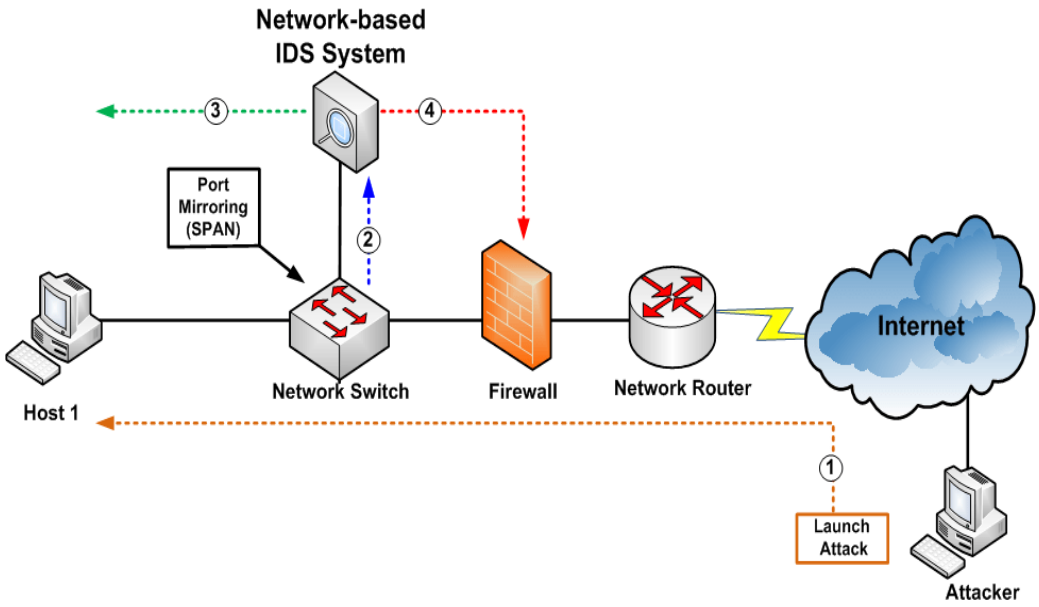

Fig 3.1 Network based IDS System

3.10 Implementation and Experiment Stage

In pursuance of building the environment and implement the testing, a range of different tools has to be used and combined like installing and configuring the environment, as well for developing the algorithms. After reviewing the tools for implementation, the following tools was chosen.

• Gym: Reinforcement algorithm libraries. More specifically this toolkit used for developing and comparing Deep Q network algorithm. The main task is to supports the teaching agents. To update the Q value we have used Gym.

• Tensorflow: Numerical computation using data flow graphs and convolution neural network building. We used tensorflow to build 4 hidden layers to compute.

• Scikit-learn: Machine learning module, an effective tool for data mining and data analysis. We have used for analysing our data.

• Python: as the scripting language for automating experiments and extracting data.

• Keras: Keras API integrates very easy with the TensorFlow workflows. It's very useful in Reinforcement Learning algorithm.

3.11 System Description

Network Intrusion Detection system (NIDS) is a system which monitors network intrusion. Intrusion may be detected by techniques like anomaly detection, signature pattern matching etc. Anomaly detection is a method in which normal network behaviour is captured and any abnormality in the network is detected such as a sudden increase in network traffic rate (number of IP packets per second). Signature pattern matching is a method in which network data is compared with the known attack techniques that are saved in a database. Intrusion is detected and system administrator is alerted about the kind of intrusion when any one of the following events takes place

• 1. If a foreign entity has been detected in a log entry.

• 2. If user tries to access information which is beyond his/her access.
• 3. Baseline for critical system resources is measured such as CPU utilization, file entries, disk activity, user logins etc. Then the system can trigger when there is a deviation from this baseline.

3.12 Development of Deep Reinforcement Learning Algorithm

Reinforcement Learning indicates a Machine Learning method in which the agent receives a delayed reward in the next time step to evaluate its previous action. Q-Learning is a value-based Reinforcement Learning algorithm. The goal is to maximize the value function "Q", which is an expected future reward given a state and action.

3.12.1 Creation of Q-Table

Create a table to calculate the maximum expected future reward, for each action in each state in the environment. Each Q-table score will be the maximum expected future reward that an agent will get if it takes an action at a state with the best policy given. To calculate the values for each element of the Q table, we use the Q learning algorithm.

3.12.2 Q-Learning Algorithm

Learning the Action Value Function the Action Value Function (or "Q-function") takes two inputs "state" and "action." It returns the expected future reward of that action at that state. Before we explore the environment, the Q-table gives the same arbitrary fixed value (most of the time 0). As we explore the environment, the Q-table will give us a better and better approximation by iteratively updating Q(s,a) using the Bellman Equation.

3.12.3 Q-Learning Algorithm Process

At the end of the training, we get "Good Q-table" with best possible results.[6] We can see this Q function as a reader that scrolls through the Q-table to find the line associated with our state, and the column associated with our action. It returns the Q value from the matching cell. This is the "expected future reward." But before we explore the environment, the Q-table gives the same arbitrary fixed value (most of the time 0). As we explore the environment, the Q-table will give us a better and better approximation by attractively updating Q(s,a) using the Bellman Equation.

At the end of the training, we get "Good Q-table" with best possible results.

1. Initialise Q- values Q(s,a) arbitrary for all state action pairs.
2. For Life or until learning is stopped.
3. Choose the action (a) in the current world state (s) based on current Q-value estimates (Q(s, .)).
4. Take the action (a) and observe the outcome state (s') and reward (r).

5. Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{\alpha} Q(S_{t+1} + \alpha) - (S_t, A_t) \right]$$

Greedy Algorithm is an algorithm which picks the best currently available option without considering the long-term effect of that decision i.e. even when the chosen decision might lead to a loss or bad long-term consequences, which may happen to be a sub optimal decision. Our approach also considers Epsilon Greedy Algorithm which helped us in gaining the optimality of the solution. It has helped us in exploring and exploiting the solution to a much further possibility in acquiring the best result.

3.12.4 Steps in designing the algorithm

- Specify Neural network Architecture
- Build Neural network
- Based on the estimated probabilities to select a random action
- Set up the training of the Neural network Using Policy Gradient

– n_iterations = 10 number of training iterations 10-100
– n_max_steps = 100 max steps per episode 100 -1000
– n_games_per_update = 10 train the policy every 10 episodes
– discount_rate = 0.95 discount rate

- Executing the graph
- Evaluation of the result

3.13 Measurement, Analysis and Comparison Stage

This is the last stage and one of the most significant task of the thesis work where all of the data is analysed. The tasks done in this stage is outlined below

- Executing the algorithm
- Analysing the obtained data and figures

3.13.1 Executing the Deep Q Algorithm

After implementation of Deep Q algorithm, during execution, following are the factors to consider Parameters

- action_index: object, reward, episode_over information
- Returns – tuple ob (object) an environment-specific object representing your observation of the environment.

– reward (float) amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
– episode_over (bool) whether it's time to reset the environment again.

– info (dict) diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change).

3.14 Analysing the Obtained Data and Figures

The implementation of the above-mentioned experiments is performed on a physical serverrunningUbuntuLinux16.04. After the implementation, the data analysis and plotting will follow.

The last stage involves analysing and verification of the data that was extracted during the experiment in-depth. The data will be analysed in an attempt to see if the objectives set in this project is met. Analysis connected to the Deep Q Learning algorithms how show well the algorithm performed by comparing the all the attributes in the detection of the network attacks.

An additional analysis involves finding the system resources (CPU, RAM) that cause the network intrusion. This consists of examining the different system resources and analyse which is likely to be the bottleneck. When performing the final evaluation and execution of the datasets (explained in the next section) with the Deep Q Learning algorithm, the execution is done over GPU's which are highly compatible with high performance.

The final analysis is to check how efficient the Deep Q Learning algorithm performs. Since the network traffic load can differ from time to time, it is important to have an analysis of the performance of the algorithm and how it reacts in response to the network traffic load.

CHAPTER 4

SOFTWARE DESIGN

4.1 UML Diagrams

4.1.1 Class Diagram

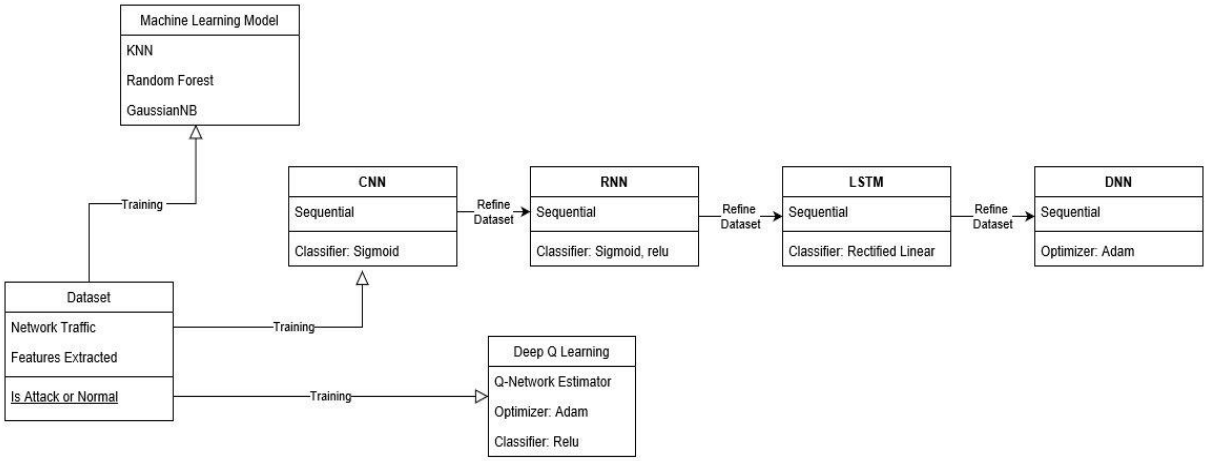

Fig 4.1 Class Diagram of NIDS

4.1.2 Use Case Diagram

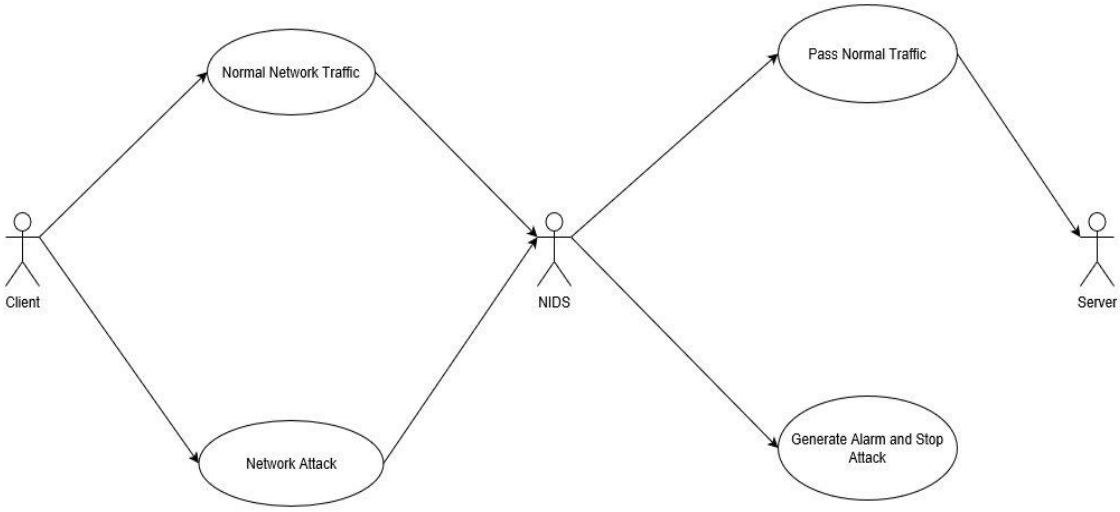

Fig 4.2 Use-Case Diagram of NIDS

4.1.3 Sequence Diagram

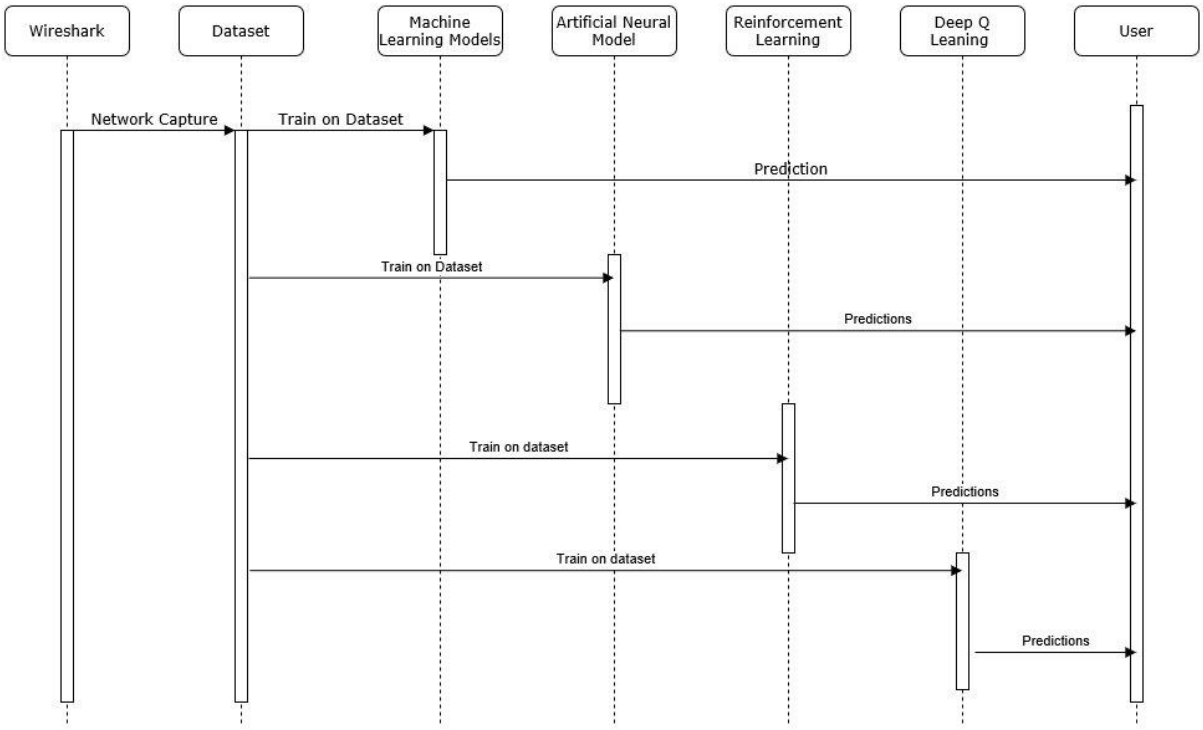

Fig 4.3 Sequence Diagram of NIDS

4.1.4 Component Diagram

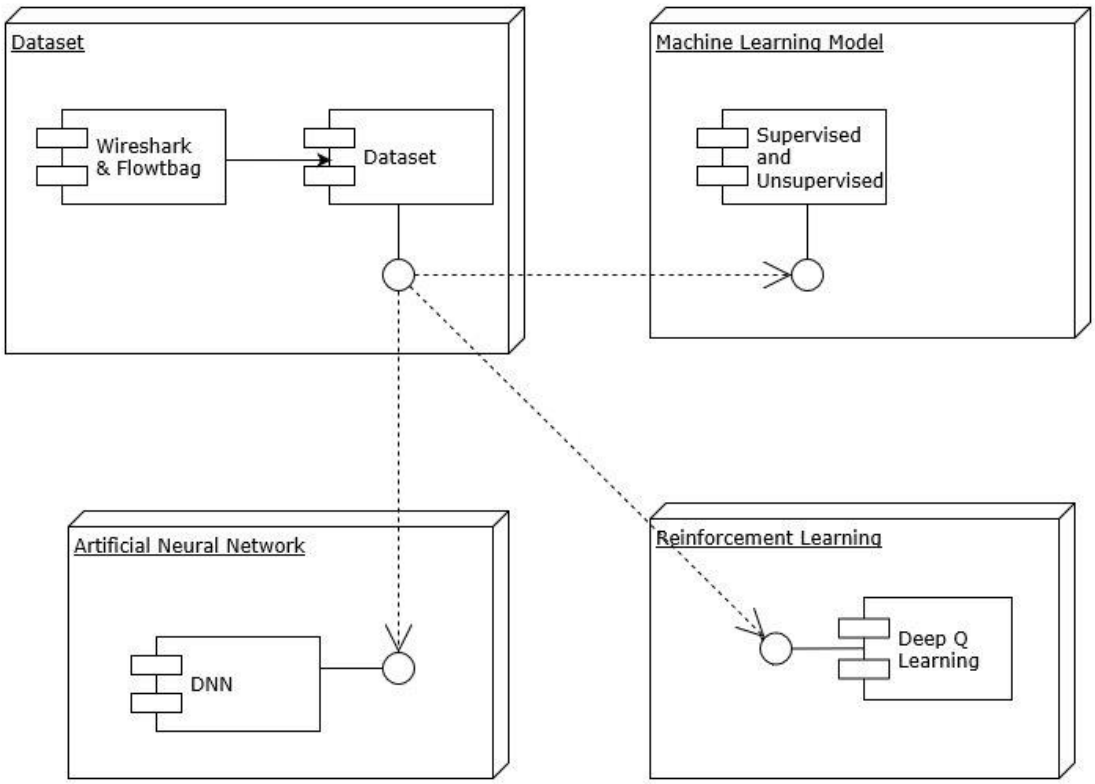

Fig 4.4 Component Diagram of NIDS

4.2 System Architecture Diagram

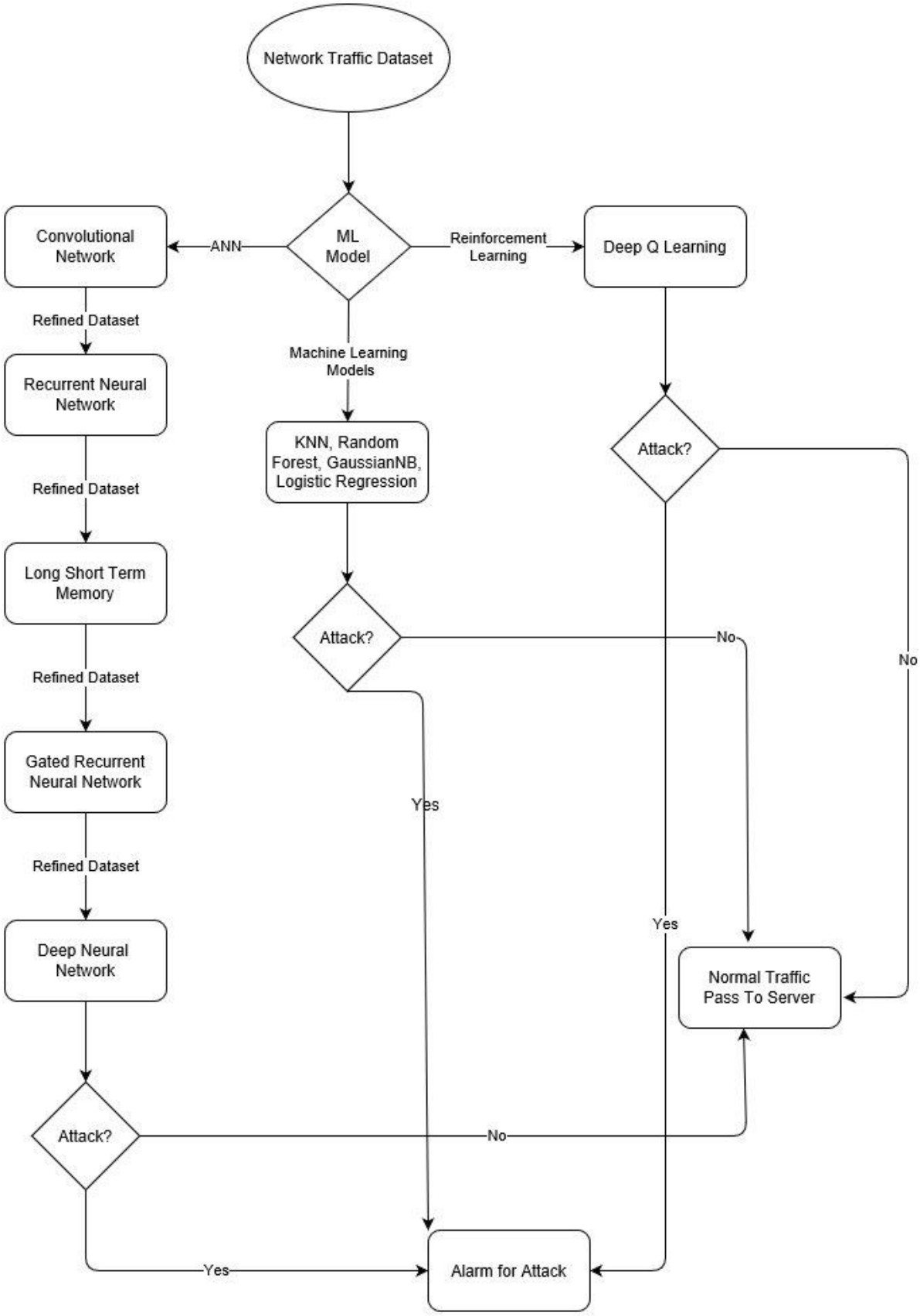

Fig 4.5 Architecture Diagram

CHAPTER 5
HARDWARE AND SOFTWARE REQUIREMENTS

5.1 Hardware Requirements

- Pentium IV or higher, (PIV-300GHz recommended)
- 256 MB RAM
- 1 Gb hard free drive space
- P-IV and Higher versions Microprocessor
- 1.44 MB FDD Disk Drives
- HP CD-Rom

5.2 Software Requirements

- Operating System: Windows 10 and Linux
- Wireshark
- Flowtbag
- Hping
- Tshark
- Python 3 and above versions
- Anaconda
- Jupyter Notebook or Spyder

5.2.1 Machine Learning Libraries and Tools

- Gym
- TensorFlow
- Sckit
- Keras
- Numpy
- Pandas
- Matplot Libraries

5.3 Description of Tools, Libraries and Software used in NIDS

Wireshark

Wireshark is an open-source packet analyser, which is used for education, analysis, software development, communication protocol development, and network troubleshooting. It is used to track the packets so that each one is filtered to meet our

specific needs. It is commonly called as a sniffer, network protocol analyser, and network analyser**.** It is also used by network security engineers to examine security problems. Wireshark is a free to use application which is used to apprehend the data back and forth. It is often called as a free packet sniffer computer application. It puts the network card into an unselective mode, i.e., to accept all the packets which it receives.

Flowtbag

The purpose of this tool is to calculate flow statistics from a given capture file. Flowtbag was designed with offline processing as the primary focus. Below are the features output by flowtbag.

- *srcip* - (*string*) The source ip address
- *srcport* - The source port number
- *dstip* - (*string*) The destination ip address
- *dstport* - The destination port number
- *proto* - The protocol (ie. TCP = 6, UDP = 17)
- *total_fpackets* - Total packets in the forward direction
- *total_fvolume* - Total bytes in the forward direction
- *total_bpackets* - Total packets in the backward direction
- *total_bvolume* - Total bytes in the backward direction
- *min_fpktl* - The size of the smallest packet sent in the forward direction (in bytes)
- *mean_fpktl* - The mean size of packets sent in the forward direction (in bytes)
- *max_fpktl* - The size of the largest packet sent in the forward direction (in bytes)
- *std_fpktl* - The standard deviation from the mean of the packets sent in the forward direction (in bytes)
- *min_bpktl* - The size of the smallest packet sent in the backward direction (in bytes)
- *mean_bpktl* - The mean size of packets sent in the backward direction (in bytes)
- *max_bpktl* - The size of the largest packet sent in the backward direction (in bytes)
- *std_bpktl* - The standard deviation from the mean of the packets sent in the backward direction (in bytes)
- *min_fiat* - The minimum amount of time between two packets sent in the forward direction (in microseconds)
- *mean_fiat* - The mean amount of time between two packets sent in the forward direction (in microseconds)
- *max_fiat* - The maximum amount of time between two packets sent in the forward direction (in microseconds)
- *std_fiat* - The standard deviation from the mean amount of time between two packets sent in the forward direction (in microseconds)
- *min_biat* - The minimum amount of time between two packets sent in the backward direction (in microseconds)
- *mean_biat* - The mean amount of time between two packets sent in the backward direction (in microseconds)

- *max_biat* - The maximum amount of time between two packets sent in the backward direction (in microseconds)
- *std_biat* - The standard deviation from the mean amount of time between two packets sent in the backward direction (in microseconds)
- *duration* - The duration of the flow (in microseconds)
- *min_active* - The minimum amount of time that the flow was active before going idle (in microseconds)
- *mean_active* - The mean amount of time that the flow was active before going idle (in microseconds)
- *max_active* - The maximum amount of time that the flow was active before going idle (in microseconds)
- *std_active* - The standard deviation from the mean amount of time that the flow was active before going idle (in microseconds)
- *min_idle* - The minimum time a flow was idle before becoming active (in microseconds)
- *mean_idle* - The mean time a flow was idle before becoming active (in microseconds)
- *max_idle* - The maximum time a flow was idle before becoming active (in microseconds)
- *std_idle* - The standard devation from the mean time a flow was idle before becoming active (in microseconds)
- *sflow_fpackets* - The average number of packets in a sub flow in the forward direction
- *sflow_fbytes* - The average number of bytes in a sub flow in the forward direction
- *sflow_bpackets* - The average number of packets in a sub flow in the backward direction
- *sflow_bbytes* - The average number of packets in a sub flow in the backward direction
- *fpsh_cnt* - The number of times the PSH flag was set in packets travelling in the forward direction
- *bpsh_cnt* - The number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
- *furg_cnt* - The number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
- *burg_cnt* - The number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
- *total_fhlen* - The total bytes used for headers in the forward direction.
- *total_bhlen* - The total bytes used for headers in the backward direction.

Hping

hping is a command-line oriented TCP/IP packet assembler/analyser. The interface is inspired to the ping(8) Unix command, but hping isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode,

the ability to send files between a covered channel, and many other features. It is used as a security tool to secure network and host.

T-shark

TShark, a well-known and powerful command-line tool and is used as a network analyser. It is developed by Wireshark. It's working structure is quite similar to Tcp dump, but it has some powerful decoders and filters. TShark is capable of capturing the data packets information of different network layers and display them in different formats. TShark is used to analyze real-time network traffic and it can read .pcap files to analyze the information, dig into the details of those connections, helping security professionals to identify their network problem.

Python 3

Python is a dynamic **o**bject-oriented programming language that can be used for many kinds of software development and other fields such as data science. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code. Python 3.0 (a.k.a. "Python 3000" or "Py3k") is a new version of the language that is incompatible with the 2.x line of releases. The language is mostly the same, but many details, especially how built-in objects like dictionaries and strings work, have changed considerably, and a lot of deprecated features have finally been removed. Also, the standard library has been reorganized in a few prominent places.

Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, which are both not free. Anaconda offers a text-mode and GUI mode, so users can install on a wide range of systems. It is designed to be easily portable and supports a wide range of hardware platforms (IA-32, Itanium, DEC_Alpha, IBM_ESA/390, PowerPC, ARMv8). It supports installing from local storage devices like CD-ROM drives and hard disks as well as from network resources via FTP, HTTP, or NFS. Installations can be automated with the use of a kickstart file, that automatically configures the installation, allowing users to run it with minimal supervision. Before starting the OS installation process, the installer checks the system hardware and

resource requirements. Only if the requirements are satisfied does it start the installation process.

Jupyter Notebook

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

Spyder

Spyder is a powerful scientific environment written in python, for python and designed by and for scientists, engineers and data analysis. It offers a unique combination of the advanced editing, analysis, debugging and profiling of a comprehensive development tool with data exploration and interactive execution, deep inspection and beautiful visualization capabilities of a scientific package. Beyond its many built in features its abilities can be extended even further via its plugin-system and API. Furthermore Spyder can also be used as a PYQt5 extension library allowing developers to build upon its functionalities and embedded its components such as interactive console in its own PYQt5 Software.

Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. The gym library is a collection of test problems — environments — that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms. Reinforcement learning (RL) is the subfield of machine learning concerned with decision making and motor control. It studies how an agent can learn how to achieve goals in a complex, uncertain environment. However, RL research is also slowed down by two factors:

- The need for better benchmarks. In supervised learning, progress has been driven by large labelled datasets like ImageNet. In RL, the closest equivalent would be a large and diverse collection of environments. However, the existing open-source collections of RL environments don’t have enough variety, and they are often difficult to even set up and use.

- Lack of standardization of environments used in publications. Subtle differences in the problem definition, such as the reward function or the set of actions, can drastically alter a task's difficulty. This issue makes it difficult to reproduce published research and compare results from different papers.

  Gym is an attempt to fix both problems.

Tensor Flow

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy. If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition. TensorFlow has always provided a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use.

Sckit

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib.The functionality that scikit-learn provides include:

- Regression, including Linear and Logistic Regression
- Classification, including K-Nearest Neighbors
- Clustering, including K-Means and K-Means++
- Model selection
- Preprocessing, including Min-Max Normalization

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result as fast as possible is key to doing good research.* Keras is the high-level API of TensorFlow 2.0: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity. Keras empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2.0: you can run Keras on TPU or on large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.

Numpy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Pandas

Pandas is used for data manipulation, analysis and cleaning. Python pandas is well suited for different kinds of data, such as:

- Tabular data with heterogeneously-typed columns
- Ordered and unordered time series data
- Arbitrary matrix data with row & column labels
- Unlabelled data
- Any other form of observational or statistical data sets

Matplot Library

matplotlib. pyplot is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits. There are several toolkits which are available that extend python matplotlib functionality. Some of them are separate downloads, others can be shipped with the matplotlib source code but have external dependencies.

CHAPTER 6
CODING/CODE TEMPLATES

6.1 Pseudo codes for Machine Learning Models

6.1.1 KNN-Model

```
model = KNeighborsClassifier()
model.fit(traindata, trainlabel)
print(model)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('res/predictedKNN.txt', predicted, fmt='%01d')
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
```

6.1.2 Linear-Regression-Model

```
model = LogisticRegression()
model.fit(traindata, trainlabel)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('res/predictedLR.txt', predicted, fmt='%01d')
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
```

6.1.3 Naïve-Base-Model

```
model = GaussianNB()
model.fit(traindata, trainlabel)
print(model)
# make predictions
expected = testlabel
```

```
predicted = model.predict(testdata)
np.savetxt('res/predictedNB.txt', predicted, fmt='%01d')
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
```

6.1.4 Decision Classifier Model

```
model = DecisionTreeClassifier()
model.fit(traindata, trainlabel)
print(model)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('res/predictedDT.txt', predicted, fmt='%01d')
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
```

6.1.5 Ada-Boost-Classifier Model

```
model = AdaBoostClassifier(n_estimators=100)
model.fit(traindata, trainlabel)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('res/predictedABoost.txt', predicted, fmt='%01d')
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
```

6.1.6 Random Forest Classifier

```
model = AdaBoostClassifier(n_estimators=100)
```

```
model.fit(traindata, trainlabel)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('res/predictedABoost.txt', predicted, fmt='%01d')
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")
 cm = metrics.confusion_matrix(expected, predicted)
 print(cm)
```

6.2 Codes for Artificial Neural Network Model

6.2.1 CNN-Model

```
lstm_output_size = 128
cnn = Sequential()
cnn.add(Convolution1D(64,3,border_mode="same",activation="relu",input_sh
ape=(41, 1)))
cnn.add(MaxPooling1D(pool_length=(2)))
cnn.add(Flatten())
cnn.add(Dense(128, activation="relu"))
cnn.add(Dropout(0.5))
cnn.add(Dense(1, activation="sigmoid"))
print(cnn.summary())
cnn.compile(loss="binary_crossentropy",
optimizer="adam",metrics=['accuracy'])
# train
checkpointer=callbacks.ModelCheckpoint(filepath="checkpoint-
{epoch:02d}.hdf5",verbose=1,save_best_only=True,
monitor='val_acc',mode='max')
csv_logger = CSVLogger('cnntrainanalysis2.csv',separator=',', append=False)
cnn.fit(X_train,y_train,nb_epoch=100,validation_data=(X_test,
y_test),callbacks=[checkpointer,csv_logger])
```

A sequential model is what we are going to use which means a feed forward model.
This layer has 128 unit.
The activation function is relu, short for rectified linear. Then we added another layer with sigmoid function for more refinement.

Now we need to compile the model. This is where we pass the setting for actually optimizing/ training the model we have defined. We are using Adam Optimizer.
Next we have loss metric. Loss is a calculation of error. A neural network doesn't actually attempt to maximise accuracy. It attempts to minimize the losses. We are using cross entropy for classification here. We will save the result in csv file and then pass the same file to next hidden layer.

6.2.2 RNN-Model

```
model = Sequential()
model.add(SimpleRNN(8,input_dim=41, return_sequences=True))
model.add(Dropout(0.1))
model.add(SimpleRNN(8, return_sequences=False)
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
checkpointer=callbacks.ModelCheckpoint(filepath="checkpoint{epoch:02d}.h
df5",verbose=1,save_best_only=True,monitor='val_acc',mode='max')
csv_logger=CSVLogger('training_set_iranalysis1.csv',separator=',',append=Fal
se)
model.fit(X_train,y_train,batch_size=batch_size,nb_epoch=100,validation_dat
a=(X_test, y_test),callbacks=[checkpointer,csv_logger])
model.save("lstm2layer_model.hdf5")
```

After Cnn model we build RNN model which is also sequential and added layers with activation function sigmoid and density 1. Now we need to compile the model. This is where we pass the setting for actually optimizing/ training the model we have defined. We are using adam optimizer in all of our ANN models. Moreover, this RNN model is trained over the refined result of CNN for more accuracy and it also has 3 hidden layers so we will pass our data through these to minimize the loss.

6.2.3 Long Short Term Memory Model

```
model = Sequential()
model.add(LSTM(8,input_dim=41, return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(8, return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(1))
```

```
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
checkpointer=callbacks.ModelCheckpoint(filepath="checkpoint{epoch:02d}.h
df5", verbose=1, save_best_only=True, monitor='val_acc',mode='max')
csv_logger=CSVLogger('training_set_iranalysis1.csv',separator=',',
append=False)
model.fit(X_train,y_train,batch_size=batch_size,nb_epoch=100,
validation_data=(X_test, y_test),callbacks=[checkpointer,csv_logger])
model.save("lstm2layer_model.hdf5")
```

This should all be feed forward so we are using sequential here. Now we are using LSTM as the layer type. The only thing is return sequence.

6.2.4 Gated Recurrent Neural Network

```
model = Sequential()
model.add(GRU(8,input_dim=41, return_sequences=True))
model.add(Dropout(0.1))
model.add(GRU(8, return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
checkpointer=callbacks.ModelCheckpoint(filepath="/checkpoint{epoch:02d}.h
df5", verbose=1, save_best_only=True, monitor='val_acc',mode='max')
csv_logger=CSVLogger('training_set_iranalysis1.csv',separator=',',
append=False)
model.fit(X_train,y_train,batch_size=batch_size,nb_epoch=1000,
validation_data=(X_test, y_test),callbacks=[checkpointer,csv_logger])
model.save("fullmodel/gru_model.hdf5")
```

In this we are using Gated Recurrent Neural Network as a layer with 8 units and return sequence as false. Activation function is sigmoid and optimizer is adam. We passed the result we get from 3 layers of LSTM to GRU. We added 2 layers of Dropout which is a regularization technique. It is a simple way to prevent overfitting.

6.2.5 Deep Neural Network Model

```
model = Sequential()
model.add(Dense(1024,input_dim=41,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(1))
```

```
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
checkpointer=callbacks.ModelCheckpoint(filepath="checkpoint-
{epoch:02d}.hdf5", verbose=1, save_best_only=True, monitor='loss')
csv_logger=CSVLogger('training_set_dnnanalysis.csv',separator=',',append=Fa
lse)
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=batch_siz
e,nb_epoch=2, callbacks=[checkpointer,csv_logger])
model.save("dnn1layer_model.hdf5")
```

A sequential model is what we are going to use which means a feed forward model. This layer has 1o24 unit. We have used a dense layer and added two layers for activation which will further refine our model those are relu and sigmoid. Output from all the layers of others ANN model will be passed into our Deep Neural Network which has also 4 hidden layers and will predict best accuracy, precision, recall and confusion metrics.

6.3 Reinforcement Learning Model

```
class RLenv(data_cls):
    def __init__(self,path,train_test,batch_size = 10,**kwargs):
        data_cls.__init__(self,path,train_test,**kwargs)
        self.batch_size = batch_size
         self.state_shape = data_cls.get_shape(self)
         def _update_state(self):
                self.states,self.labels = data_cls.get_batch(self,self.batch_size)
```

It is the defination of the Reinforcement Learning Environment.

```
def reset(self):
        self.state_numb = 0
        self.states,self.labels = data_cls.get_batch(self,self.batch_size)
        self.total_reward = 0
        self.steps_in_episode = 0
        return self.states.values
```

It returns the observation of the environment

```
def act(self,actions):
        self.reward = np.zeros(self.batch_size)
        for indx,a in enumerate(actions):
                if a == self.labels[indx]:
                self.reward[indx] = 1
```

Network Architecture of the RL

```
        model = Sequential()
        model.add(Dense(hidden_size, input_shape=(env.state_shape[1]-1,),
        batch_size=batch_size, activation='relu'))
    model.add(Dense(hidden_size, activation='relu'))
    model.add(Dense(num_actions))
    model.compile(sgd(lr=.2), "mse")

    reward_chain = []
    loss_chain = []
      self._update_state()
      self.done = False

      Get new state and new true values, Done always false in this continuous task

        return self.states, self.reward, self.done

        The above code returns the current state and the reward.

        valid_actions = [0, 1]
        num_actions = len(valid_actions)
        epsilon = .1  # exploration
         num_episodes = 300
        iterations_episode = 100

        Setting 1 for the attack and 0 for no attack

        apply actions, get rewards and new state
        next_states, reward, done = env.act(actions)

         If the epoch*batch_size*iterations_episode is largest than the df

         if next_states.shape[0] != batch_size:

        q_prime = model.predict(next_states)
        indx = np.argmax(q_prime,axis=1)
        sx = np.arange(len(indx))

        We will Update the q values.

        targets = reward + gamma * q[sx,indx]
        q[sx,actions] = targets
```

```
            After this we will train the network and update the loss
            loss += model.train_on_batch(states, q)
```

6.4 Deep Q Learning

```
class QNetwork():
    """
    Q-Network Estimator
    Represents the global model for the table
    """

    def __init__(self,obs_size,num_actions,hidden_size = 100,
                 hidden_layers = 1,learning_rate=.2):
        """
        Initialize the network with the provided shape
        """
        self.obs_size = obs_size
        self.num_actions = num_actions

        # Network arquitecture
        self.model = Sequential()
        # Add imput layer
        self.model.add(Dense(hidden_size, input_shape=(obs_size,),
                             activation='relu'))
        # Add hidden layers
        for layers in range(hidden_layers):
            self.model.add(Dense(hidden_size, activation='relu'))
        # Add output layer
        self.model.add(Dense(num_actions))

        #optimizer = optimizers.SGD(learning_rate)
        # optimizer = optimizers.Adam(alpha=learning_rate)
        optimizer = optimizers.Adam(0.00025)
        # optimizer = optimizers.RMSpropGraves(learning_rate, 0.95, self.momentum,
1e-2)

        # Compilation of the model with optimizer and loss
        self.model.compile(loss=huber_loss,optimizer=optimizer)

    def predict(self,state,batch_size=1):
        """
        Predicts action values.
        """
        return self.model.predict(state,batch_size=batch_size)
```

```
    def update(self, states, q):
        """
        Updates the estimator with the targets.

        Args:
          states: Target states
          q: Estimated values

        Returns:
          The calculated loss on the batch.
        """
        loss = self.model.train_on_batch(states, q)
        return loss

    def copy_model(model):
        """Returns a copy of a keras model."""
        model.save('tmp_model')
        return keras.models.load_model('tmp_model')

'''
Reinforcement learning Agent definition
'''

class Agent(object):

    def __init__(self, actions,obs_size, policy="EpsilonGreedy", **kwargs):
        self.actions = actions
        self.num_actions = len(actions)
        self.obs_size = obs_size

        self.epsilon = kwargs.get('epsilon', 1)
        self.min_epsilon = kwargs.get('min_epsilon', .1)
        self.gamma = kwargs.get('gamma', .001)
        self.minibatch_size = kwargs.get('minibatch_size', 2)
        self.epoch_length = kwargs.get('epoch_length', 100)
        self.decay_rate = kwargs.get('decay_rate',0.99)
        self.ExpRep = kwargs.get('ExpRep',True)
        if self.ExpRep:
            self.memory = ReplayMemory(self.obs_size, kwargs.get('mem_size', 10))

        self.ddqn_time = 100
        self.ddqn_update = self.ddqn_time
```

```
        self.model_network = QNetwork(self.obs_size, self.num_actions,
                                      kwargs.get('hidden_size', 100),
                                      kwargs.get('hidden_layers',1),
                                      kwargs.get('learning_rate',.2))
        self.target_model_network = QNetwork(self.obs_size, self.num_actions,
                                      kwargs.get('hidden_size', 100),
                                      kwargs.get('hidden_layers',1),
                                      kwargs.get('learning_rate',.2))
        self.target_model_network.model                                    =
QNetwork.copy_model(self.model_network.model)

        if policy == "EpsilonGreedy":
            self.policy = Epsilon_greedy(self.model_network,len(actions),
                                         self.epsilon,self.min_epsilon,
                                         self.decay_rate,self.epoch_length)

    def learn(self, states, actions,next_states, rewards, done):
        if self.ExpRep:
            self.memory.observe(states, actions, rewards, done)
        else:
            self.states = states
            self.actions = actions
            self.next_states = next_states
            self.rewards = rewards
            self.done = done
       def update_model(self):
        if self.ExpRep:
            (states,          actions,          rewards,          next_states,          done)          =
self.memory.sample_minibatch(self.minibatch_size)
        else:
            states = self.states
            rewards = self.rewards
            next_states = self.next_states
            actions = self.actions
            done = self.done

        next_actions = []
        # Compute Q targets
#         Q_prime = self.model_network.predict(next_states,self.minibatch_size)
        Q_prime = self.target_model_network.predict(next_states,self.minibatch_size)
        # TODO: fix performance in this loop
        for row in range(Q_prime.shape[0]):
            best_next_actions = np.argwhere(Q_prime[row] == np.amax(Q_prime[row]))
```

```
next_actions.append(best_next_actions[np.random.choice(len(best_next_actions))].ite
m())
        sx = np.arange(len(next_actions))
```

CHAPTER 7
TESTING

7.1 Introduction

Testing is process of ensuring that software function as per user needs.
Accuracy: Defined as the percentage of correctly classified records over the total number of records. This is the percentage of correctly classified item over total item. In our code we have done like this

accuracy =(tp + tn)/totalrecords

- tp=true positive
- tn=true negatives

Precision is the ratio of true positive value and the combination of true positive value and false positive.

precision = tp / (tp + fp)

Sensitivity: Its also called recall. This is the ration of true positive and sum of true positive and false negative.

Specificity: This is the ration of true negative and sum of true negative and false positive.

specificity = tn / (tn + fp)

7.2 Types of software testing:

1. White Box Testing It is a test case design philosophy that uses the control structure described as part of component level design to derive test cases.
   • Basis Path Testing- Basis path method enables designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.
   • Contol Structure Testing- It tests control structures of program.

2. Black Box Testing Black box testing enables the software designer to derive sets of input conditions that will fully exercise all functional requirements of a program.
   • Graph Based Testing Method
   • Equivalence Partitioning
   • Boundary Value Analysis
   • Orthogonal Array Testing

3. Integration Testing
   Integration Testing is a systematic technique for constructing software architecture while at the same time conducting tests to uncover errors associated with interfacing.

4. Regression Testing

Each time a new model is added as part of integration testing, the software changes. At this time regression testing is applied.

7.3 Testing on KDD Dataset of Network Intrusion

We have trained our algorithms on the dataset which we have captured from Wireshark in which we have done DoS attack using hping3 and then we extracted 32 features using Flowtbag tool and hence we made a dataset which is equal to KDD dataset in terms of number of features. So now its time to test our algorithms on KDD dataset but our model is trained with our dataset only.

7.3.1 KDDCUP Dataset

Computer network intrusion detection, this database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. Intelligent Intrusion detection systems can only build if there is an availability of a valid data set. A data set with the sizable amount of data which mimics the real-time can only help to train and test an intrusion detection system.

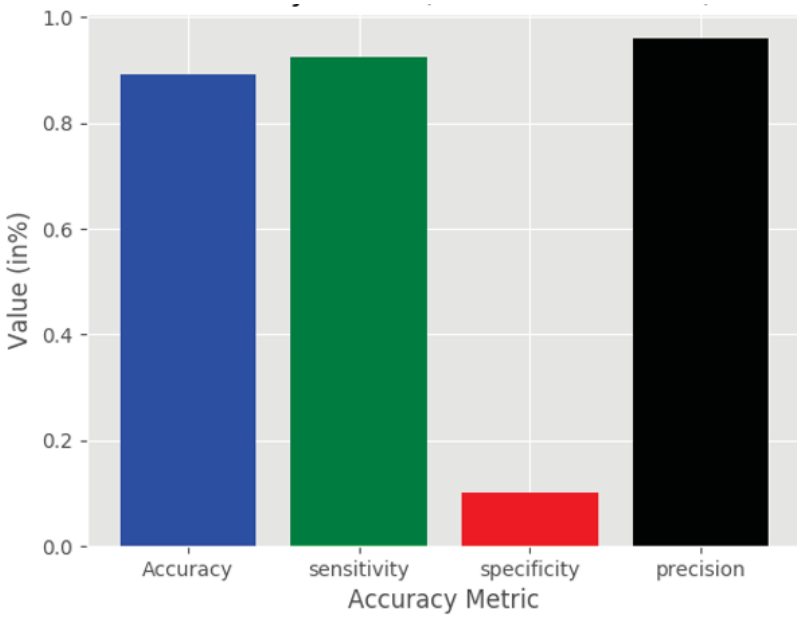

Fig 7.1 Accuracy Metric for KDD

Result using KDD-Dataset with Machine Learning-Model (Random Forest)

- Accuracy = 89.245%
- Precision=95.93%
- Sensitivity=92.34%
- Specificity=10.1%

In the result above is the confusion matrix including the value of true posititve and true false with true false and true positive. Then accuracy of predicting the attack with its precision and its recall value and F_score.

7.4 Testing on NSL-KDD Dataset

7.4.1 NSL-KDD

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 data set. Furthermore, the number of records in the NSL-KDD train and test sets are reasonable. This advantage makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research work will be consistent and comparable. The NSL-KDD data set has the following advantages over the original KDD data set:

- It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.
- There is no duplicate records in the proposed test sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on the frequent records.
- The number of selected records from each difficultylevel group is inversely proportional to the percentage of records in the original KDD data set. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques.
- The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research works will be consistent and comparable.

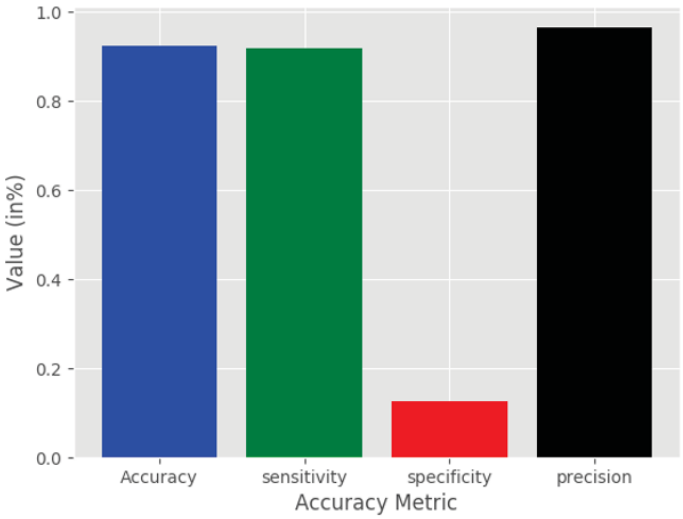

Fig 7.2 Accuracy Metric for NSL-KDD

Result using NSL-KDDCup Dataset using ANN

- Accuracy = 92.32%
- Precision=96.54%
- Sensitivity=93.56%
- Specificity=10.35%

7.5 Testing on CICIDS 2017 Dataset

7.5.1 CICIDS Dataset

KDDcup99 is 20 years older dataset and NSL-KDD is 10 years older. Network packet attributes have changed and are upgraded. KDDcup99 has only 41 attributes. CICIDS 2017 dataset is used which has 85 attributes. The size and entry of the dataset are 20 times more that KDDcup99 or NSL-KDD. Moreover, tuning the hyperparameters could improve the rate of accuracy because the prediction of the model relies on the dataset.

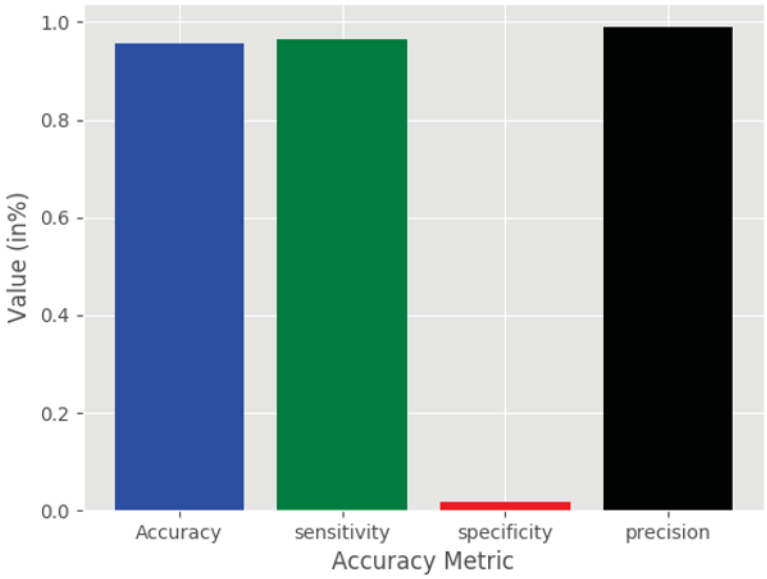

Fig 7.3 Accuracy Metric for CICIDS

Result using Deep Q learning Model on CICIDS Dataset

- Accuracy = 95.53%
- Precision=98.94%
- Sensitivity=96.51%
- Specificity=1.7%

We have evaluated three datasets. We got most consistent precision and sensitivity performance. Using DQN our highest outcome percentage is almost 99% precision and sensitivity is more than 96%. Our result reflects thatDQN is best suitable for unknown attacks detection compare to other ML algorithm and minimise the false alarm rate. We have used

DQN for detection unknown attacks. While testing our model with new dataset, the input is random and unknown to the model. Using DQN we have got the maximum accuracy 95% and precision of 98%. We have shown the comparison with other algorithm and got acceptable result according to other algorithm.

CHAPTER 8
OUTPUT SCREENS

8.1 KNN- Output

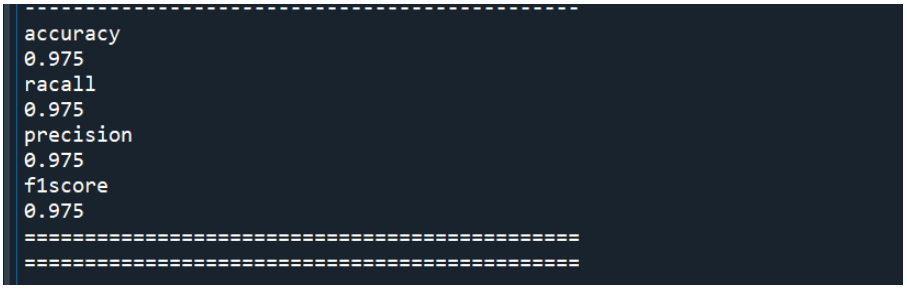

Fig 8.1 Output Screen of KNN-Algorithm

8.2 Logistic Regression Output

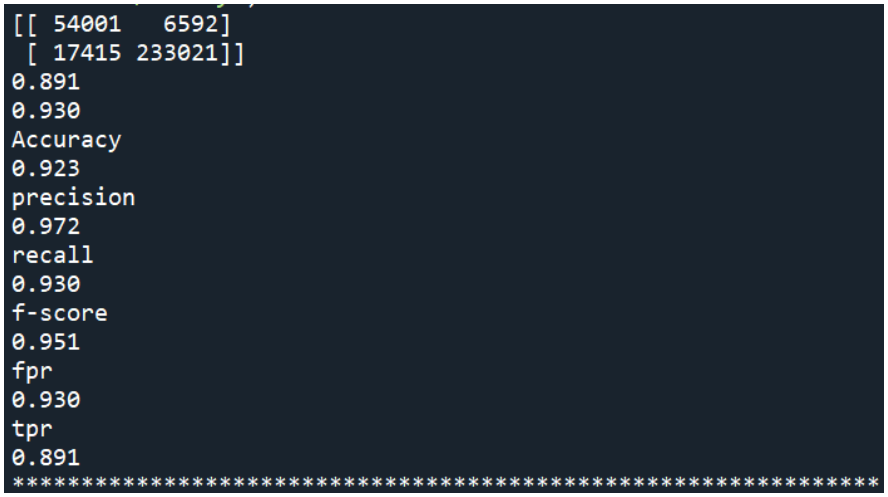

Fig 8.2 Output Screen of Logistic Regression-Algorithm

8.3 Random Forest Output

```
[[ 59755    838]
 [ 21845 228591]]
0.986
0.913
Accuracy
0.927
precision
0.996
recall
0.913
f-score
0.953
fpr
0.913
tpr
0.986
**********************************************************
```

Fig 8.3 Output Screen of RF-Algorithm

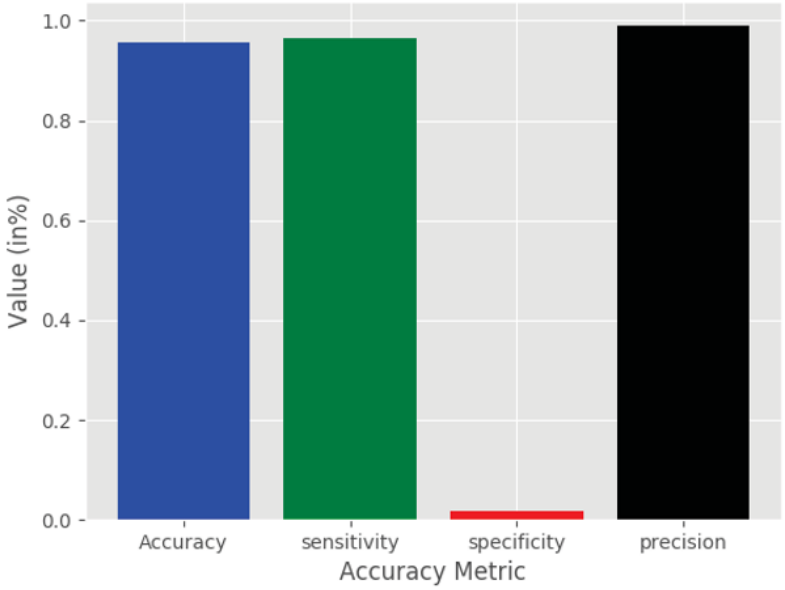

Fig 8.4 Accuracy Metric of RF-Algorithm

8.4 Decision Tree Output

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
[[ 59736    857]
 [ 20287 230149]]
0.986
0.919
Accuracy
0.932
precision
0.996
recall
0.919
f-score
0.956
fpr
0.919
tpr
0.986
************************************************************
```

Fig 8.5 Output Screen of Decision Tree-Algorithm

8.5 GaussianNB Output

```
GaussianNB(priors=None, var_smoothing=1e-09)
[[ 54485   6108]
 [ 21009 229427]]
0.899
0.916
Accuracy
0.913
precision
0.974
recall
0.916
f-score
0.944
fpr
0.916
tpr
0.899
************************************************************

In [3]:
```

Fig 8.6 Output Screen of GaussianNB-Algorithm

8.6 Artificial Neural Network Outputs

8.6.1 CNN Output

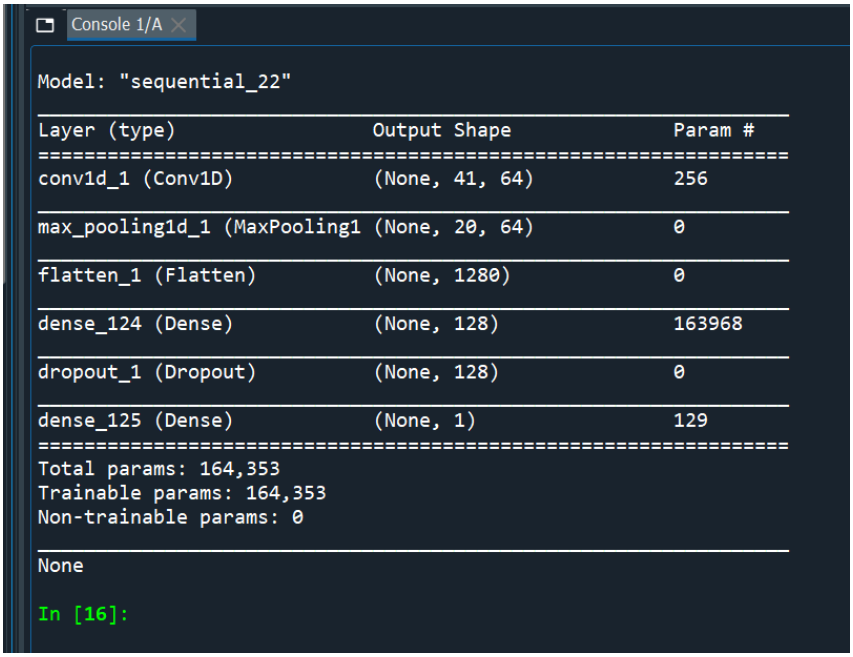

Fig 8.7 After Training the machine on CNN

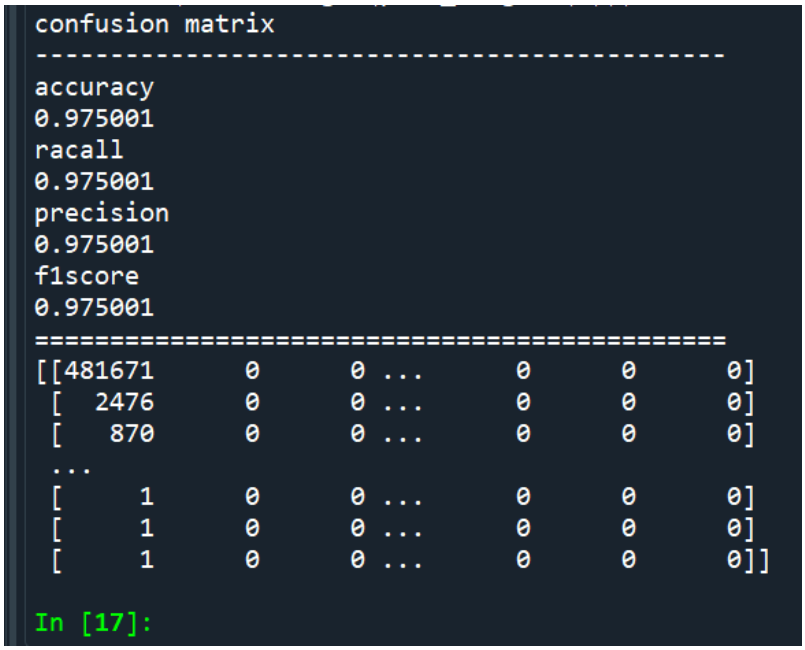

Fig 8.8 Confusion Matrix, Accuracy, Precision and F1_score of CNN-Algorithm

8.6.2 RNN Output

```
Train on 494021 samples, validate on 494021 samples
Epoch 1/1
494021/494021 [==============================] - 147s 297us/step - loss: -1290.4920 - accuracy:
0.7337 - val_loss: -2432.3705 - val_accuracy: 0.8083 4:04 - loss: -20.6852 - accuracy: 0.0372
79520/494021 [===>..........................] - ETA: 1:37 - loss: -282.9034 - accuracy: 0.4058
C:\Users\vaibhav\anaconda31\lib\site-packages\keras\callbacks\callbacks.py:707: RuntimeWarning: Can
save best model only with val_acc available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
494021/494021 [==============================] - 46s 93us/step

Loss: -2432.37, Accuracy: 80.83%

In [21]:
```

Fig 8.9 Training Output

```
confusion matrix
-----------------------------------------------
accuracy
0.975001
racall
0.975001
precision
0.975001
f1score
0.975001
===============================================
[[481671      0      0 ...      0      0      0]
 [  2476      0      0 ...      0      0      0]
 [   870      0      0 ...      0      0      0]
 ...
 [     1      0      0 ...      0      0      0]
 [     1      0      0 ...      0      0      0]
 [     1      0      0 ...      0      0      0]]

In [19]:
```

Fig 8.10 Output of RNN

8.6.3 LSTM Output

```
Epoch 2/3
   640/494021 [..............................] - ETA: 1:55 - loss: -2735.8080 - accuracy: 0.8141C:
\Users\vaibhav\anaconda31\lib\site-packages\keras\callbacks\callbacks.py:707: RuntimeWarning: Can
save best model only with val_acc available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
494021/494021 [==============================] - 171s 346us/step - loss: -2913.0070 - accuracy:
0.8048 - val_loss: -3801.2018 - val_accuracy: 0.8076: 1:55 - loss: -1857.4683 - accuracy:
0.8058159840/494021 [========>.....................] - ETA: 1:29 - loss: -2176.5489 - accuracy:
0.8036234464/494021 [=============>................] - ETA: 1:09 - loss: -2312.2814 - accuracy:
0.8045283360/494021 [================>.............] - ETA: 56s - loss: -2477.5135 - accuracy:
0.8048
Epoch 3/3
494021/494021 [==============================] - 182s 368us/step - loss: -4688.2336 - accuracy:
0.8085 - val_loss: -5611.0405 - val_accuracy: 0.8097: 2:01 - loss: -3941.7205 - accuracy:
0.8094488160/494021 [============================>.] - ETA: 1s - loss: -4672.7969 - accuracy:
0.8086
494021/494021 [==============================] - 35s 70us/step

Loss: -5611.04, Accuracy: 80.97%
```

Fig 8.11 Output of LSTM

8.6.4 GRU Output

```
[[ 54001   6592]
 [ 17415 233021]]
0.891
0.930
Accuracy
0.923
precision
0.972
recall
0.930
f-score
0.951
fpr
0.930
tpr
0.891
**********************************************************************
```

Fig 8.12 Output of GRU

8.6.5 Deep Neural Network Output

Fig 8.13 Graph of TP, TN, FP, FN after ANN

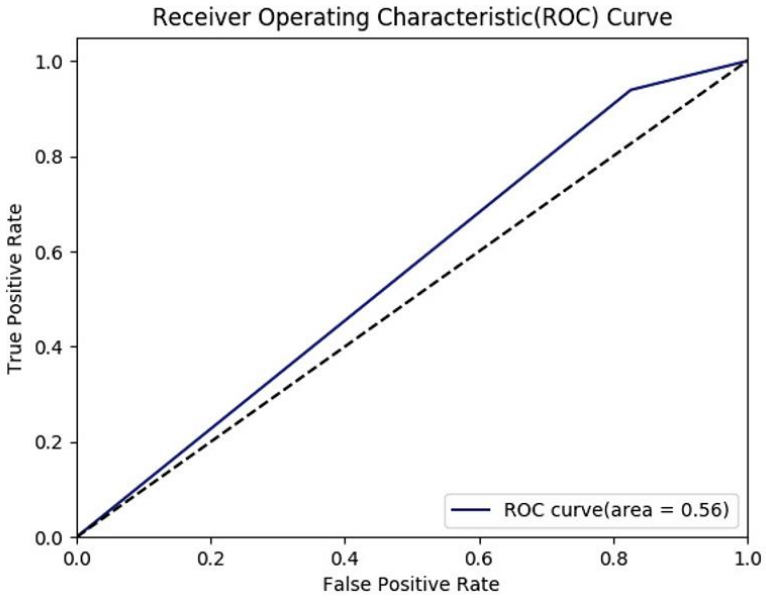

Fig 8.14 ROC CURVE after complete ANN Model

8.7 Reinforcement Learning Output

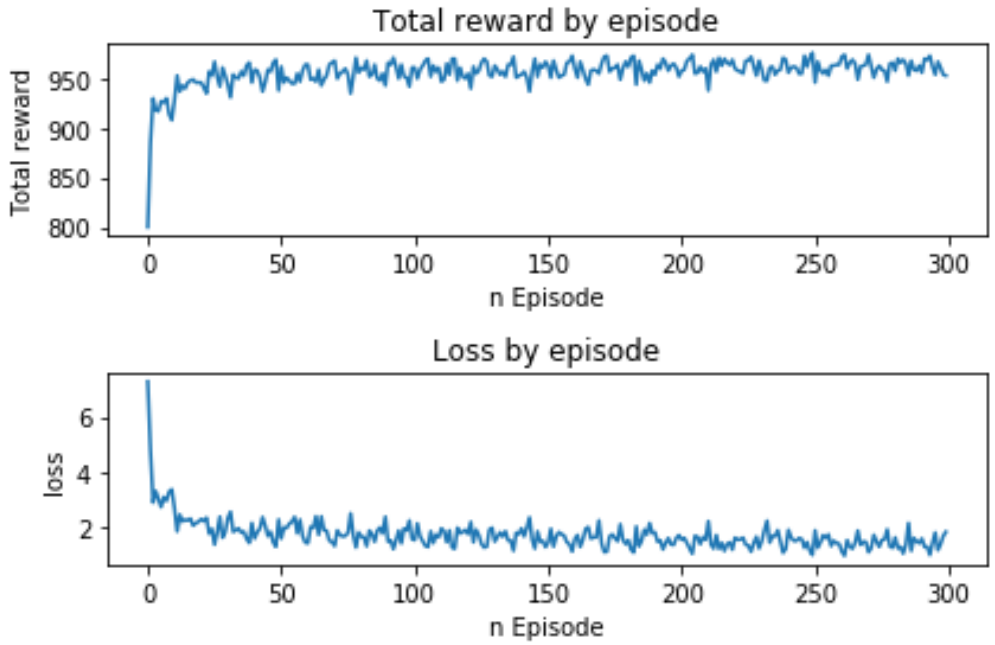

Fig 8.15 Total Reward and Loss by n Episodes

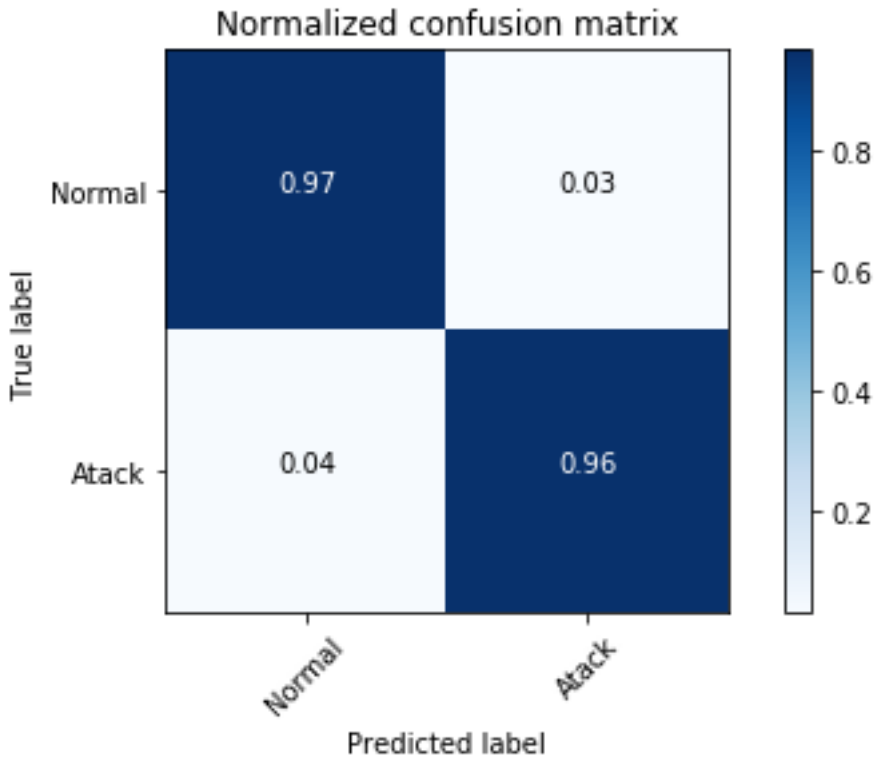

Fig 8.16 Confusion Matrix of RL-Algorithm

8.8 Deep Q Learning Output

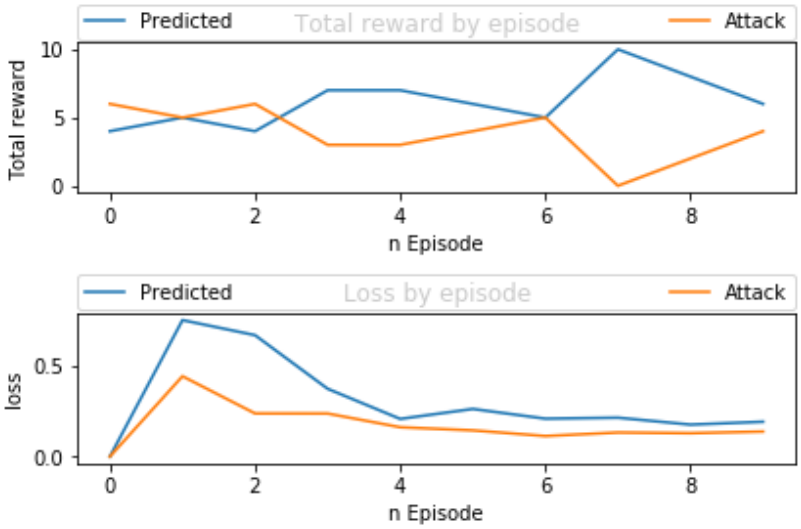

Fig 8.17 Reward and Loss for 10 Iterations

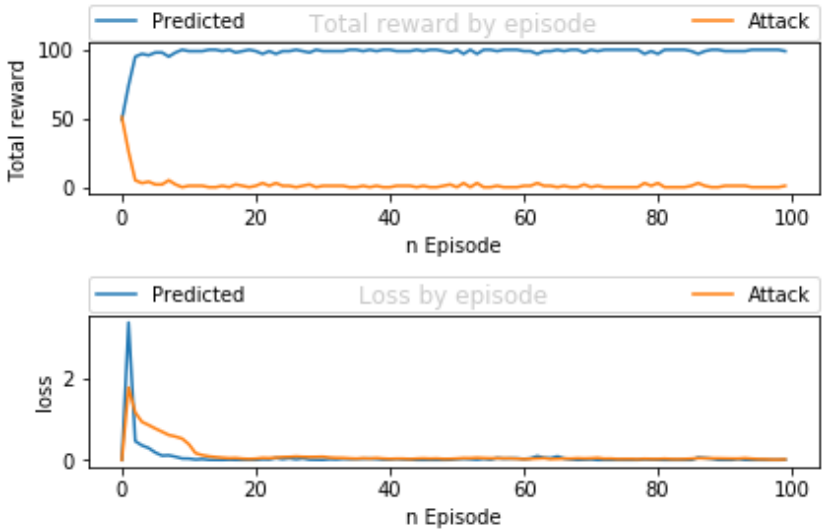

Fig 8.18 Reward and Loss for 100 Iterations

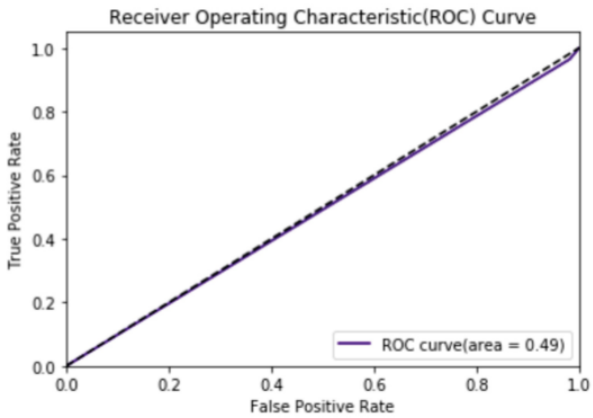

Fig 8.19 ROC Curve for Deep Q learning

CHAPTER 9
CONCLUSION

Prevention of security breaches completely using the existing security technologies is unrealistic. As a result, intrusion detection is an important component in network security. IDS offers the potential advantages of reducing the man power needed in monitoring, increasing detection efficiency, providing data that would otherwise not be available, helping the information security community learn about new vulnerabilities and providing legal evidence.

In this system we proposed a new intrusion detection system in which we have refined our results by combining different artificial neural networks having 3-4 hidden layers to get the best accuracy for predicting the attacks which are already detected. We trained our model first with CNN and passed the result in RNN then GRU through LSTM and at last passed the refined dataset to Deep Neural Network with 5 hidden layers. We achieved better accuracy after this combined model than using a single model for the detection but still we faces the problem of unknown attacks detection and older dataset problem so we used reinforcement learning which is model-free approach of training the machine on new datasets and can also predict unknown attacks.

We also examined that the Deep Q Network is able to classify the attack types with a highest accuracy of 97%. In opposition, there are other algorithms we have shown that has high accuracy level. Moreover, tuning the hyperparameters could improve the rate of accuracy because the prediction of the model relies on the dataset. The Deep Q network algorithm provides a very good accuracy compared with other models with a low featured dataset. For all the research and experimentation of NIDS with machine learning algorithm approach, these datasets are used for Q network and hence we are comparing our result with these available models on the internet. In application, this model could be implemented for the detection of any unknown attack and unknown data type. One of the most important concern to consider is that to make sure the data is clean. From the results and analysis, Deep Reinforcement Learning algorithm with NIDS would be a fair consideration for the enhancement of the improved accuracy of intrusion detection in the network with high performance metrics. On the other hand, prior to the known attacks, we can also detect the unknown attacks. Deep Reinforcement Learning techniques help in acquiring the best possible outcome with better accuracy i.e. the maximum reward. Advantage of Reinforcement Learning technique is doesn’t need any prior knowledge or past experience to take any decision, when the possibility of occurrence of anomaly packets is high; when there is an unknown attack, it also detects random data.

CHAPTER 10
FURTHER ENHANCEMENTS AND RECOMMENDATION

In this section, we propose further improvements and research opportunities following from this research.

• Working on distributed tensorflow to use multiple GPUs which will reduce the run time.

• Extending the initial work on anomaly protocol intrusion detection using TCP and covering other communication protocols.

• NIDS uses network packets as source of input. The packet structure covers all the layers of the network including the application layer. Thus, writing application specific intrusion detection is further extension to this research. A research opportunity is a study which deals with the specified types of attacks in our research with the proposed NIDS system.

• Further, evaluation of TeStID is to install it on an appliance and compare it to appliance based IDSs.

• Using other techniques with SDP for intrusion detection can be investigated such as using neural networks, fuzzy logic, probabilistic logic, etc.

• Detecting intrusions in encrypted traffic. Formulating specifications in temporal logic from normal traffic for application or Protocol.

CHAPTER 11

REFERENCES

[1] A. M. Chandrasekhar, K. Raghuveer " Intrusion Detection Techniques by using K-means, Fuzzy Neural network and SVM classifier", ICCCI2013, Jan. 04-06, 2013, Coimbatore, INDIA.

[2] Sumaiya Thaseen, Ch. Aswani Kumar: "analysis of supervised Tree based classifiers for intrusion detection system", In Proceedings of the 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME) in February 21-22.

[3] Nidhi Srivastav, Rama Krishna Challa "Novel Intrusion Detection System integrating Layered Framework with Neural Network" 978-1-4673- 4529, in IEEE, 2012.

[4] Zhao Y ongli, Zhang Yungui, Tong Weiming, Chen Hongzhi, "an improved feature selection algorithm based on MAHALANOBIS distance for network intrusion detection", international conference on sensor network security technology and privacy communication system (SNS & PCS), 2013.

[5] . Fengli Zhang, Dan Wang "an effective feature selection approach for network intrusion detection" IEEE Eighth international conference on networking, architecture and storage, 2013.

[6] Y. Li, B. Fang, Y. Chen, and L. Guo, "a lightweight intrusion detection model based on feature selection and maximum entropy model," Proc. International Conference on Communication Technology (ICCT '06), IEEE Press, Nov. 2006, pp.1-4, doi: 10.1109/ICCT.2006.341771.

[7] . Preecha Somwang, Woraphon Lilakiatsakun: "intrusion detection technique by using fuzzy ART on computer network security" 2011, 978-1- 4577-2119, in IEEE.

[8] Bauer, D.S., Eichelman II., F.R., Herrera, R.M. and Irgon, A.E. (1989) 'Intrusion detection: an application of expert systems to computer security', Proceedings of the International Carnahan Conference on Security Technology, pp. 97-100.

[9] A novel intrusion detection system based on hierarchical clustering and support vector machines SJ Horng, MY Su, YH Chen, TW Kao, RJ Chen, JL Lai… - Expert systems with Applications, 2011.

[10] Richard, K. and V. Giovanni (2002). Intrusion Detection: A Brief History and Overview. Security and Privacy: 27-30.

[11] Adebayo, O.A., S.O. Falaki, O.S. Adewale and Boniface K. Alese, 2008.. Network Intrusion Detection Based On Rough Set and K-Nearest Neighbour.

[12] Abadeh, M. S., Mohamadi, H. and Habibi, J. "Design and analysis of genetic fuzzy systems for intrusion detection in computer networks", Expert Systems with Applications, 38(6), pp. 7067–7075, 2011.

[13] Intrusion detection using fuzzy association rules A Tajbakhsh, M Rahmati, A Mirzaei - Applied Soft Computing, 2009