University Mohamed Khider of Biskra

Intelligent Network Intrusion Detection Systems

by

Labed Abdel Djalil

Exact Sciences and Natural and Life Sciences
Computer Science

June 2018

University Mohamed Khider of Biskra

Abstract

Exact Sciences and Natural and Life Sciences
Computer Science

Master Thesis

by Labed Abdel Djalil

Network Intrusion Detection System (NIDS) is a security mechanism used to protect a computer network from malicious activity and unauthorized access to devices by generating reports to the administrator of the system. In our project, we propose a deep learning approach for intrusion detection using a deep neural network (DNN-IDS). We apply our proposal on NSL-KDD (a benchmark dataset for network intrusion). We particularly study the performance of the model with both binary and multiclass classifications. The binary classification model aims at identifying whether network traffic behaviour is normal or anomalous whereas multiclass classification provides a more refined classification by determining the class to which traffic belongs (Normal, Denial-of-Service, User to Root, Probe, Remote to Local). We validate our proposal by evaluating its performance against state-of-the-art machine learning classification methods in both situations: binary and multiclass classification. The experimental results show that our approach performs very well compared to existing NIDSs.

University Mohamed Khider of Biskra

Résumé

Exact Sciences and Natural and Life Sciences
Computer Science

Mémoire de Master

par Labed Abdel Djalil

Un système de détection d'intrusion réseau est un système de sécurité utilisé pour protéger un environnement réseau contre les activités malveillantes et l'accès non autorisé aux périphériques en générant des rapports pour l'administrateur du système. Dans notre projet, nous proposons une approche d'apprentissage en profondeur pour la détection d'intrusion à l'aide d'un réseau neuronal profond (DNN-IDS). Nous applicons notre proposition sur NSL-KDD (un ensemble de données de référence pour l'intrusion réseau). De plus, nous étudions les performances du modèle avec des classifications binaire et multiclasse. Le modèle de classification binaire identifie si le comportement du trafic réseau est normal ou anormal. Le modèle de classification multiclasse vise classer le traffic dans une des cinq catégories suivantes: (Normal, Deni de Service, User to Root, Probe, Remote to Local). Pour valider notre proposition, nous la comparons avec les méthodes de pointe de classification d'apprentissage automatique dans les deux situations: classification binaire et multiclasse. Les résultats des expérimentations ont prouvé que l'approche proposée se comporte très bien par rapport aux NIDSs précédemment mis en oeuvre.

Acknowledgements

The first and the last thing is for Allah Who Provided me the sufficient capacity to finish this work.

I would like to express my thanks for the great support from many people, without whom this thesis would not have been possible.

I would like to express uttermost gratitude to my supervisor Prof Abdelmalik Bachir for his guidance, advice and effort, which have been essential at every stage of my research and shaped the thesis.

Also I am very thankful to my friends Labed Ayoub, Labed Fathi, Labed Nadir, Labed Kamel, Rabie Bouchamie, Labed Saleh, Labed Taher, Labed Mohamed Mohyeddine, Ben Nacer Mostafa, Hadef Mehdi, Redjimi Adel, Noureddine Houssem Eddine, Kherachi Fadia, Amina Rouina, Samah Merabti, Labed Razgallah, Ben Terki Aboubaker Seddiq, EL Hamel Soheib, Ben Hammed Radoune for the friendship, inspiration, and encouragement during my research.

Most importantly, I would like to thank my mother Labed Samia, my father Mansouf, my grandpa Azzedine and grandma Bachra for their constant support, encouragement, and motivation, which enabled me to overcome any difficulties I encountered during my research

At last thanks go to my family members Mohamed, Hadjer and Rinad for their companion and the great happiness they brought.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
CNN	**C**onvolutional **N**eural **N**etwork
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
DNN	**D**eep **N**eural **N**etwork
DMZ	**D**e**M**ilitarized **Z**one
DoS	**D**enial **O**f **S**ervice
HIDS	**H**ost based **I**ntrusion **D**etection **S**ystem
GD	**G**radient **N**eural **D**esecent
IPS	**I**ntrusion **P**revention **S**ystem
IDS	**I**ntrusion **D**etection **S**ystem
LAN	**L**ocal **A**rea **N**etwork
MLE	**M**aximum **L**ikelihood **E**stimation
MSE	**M**ean **S**equard **E**rror
NIDS	**N**etwork based **I**ntrusion **D**etection **S**ystem
NLP	**N**atural **L**anguage **P**rocessing
R2L	**R**oot **T**o **L**ocal
RBM	**R**estricted **B**oltzman **M**achines
ReLU	**R**ectified **L**inear **U**nits
RMSE	**R**oot **M**ean **S**equard **E**rror
RNN	**R**ecurrent **N**eural **N**etwork
SGD	**S**tochasitc **G**radient **D**esecent
SVM	**S**upport **V**ector **M**achine
U2R	**U**ser **T**o **R**oot

Introduction

In the age where everything becomes connected and the increasingly deep integration of the internet and society, the internet is changing the way in which people live, study, and work. The importance of the security measures grows bigger as the protection of data (e.g., company data and research data) or the protection against intrusion attacks (virus, worm, Trojan horse, DoS etc...) are becoming more and more frequent for almost everyone working with a connected machine. These attacks are used to illegally gain access to unauthorized information, misuse of information, or to reduce the availability of the information to authorized users.

The work concerns a certain method for protecting computer networks. Network Intrusion Detection System (NIDS) is one way of protecting a computer network. This kind of technology enables users of a network to be aware of the incoming threats from the Internet by observing and analyzing network traffic. They collect and check packets, looking for a malicious packet and their behaviors. As soon as a suspicious event is detected, security process takes action to warn the administrator by writing out to log files, which the administrator can read and discover possible intrusion. The IDS does not prevent an intrusion like a firewall which closes ports entirely. The IDS lets the traffic flow but sees the traffic and detects intrusion without really doing anything about it. The rest is up to the administrator or the security policy. Usually an IDS is based on anomaly detection by making use of different detection techniques such as machine learning which has proved its effectiveness in this field.

Machine learning methodologies have been widely used in identifying various types of attacks, and a machine learning approach can help the network administrator take the corresponding measures for preventing intrusions. The NIDSs are developed as classifiers to differentiate the normal traffic from the anomalous traffic. Recently, deep learning

based methods have been successfully applied in audio, image, and speech processing applications. Because of the success of deep learning, we have proposed to investigate its use for an intrusion detection system using deep neural networks. Our method called DNN-IDS can operate on two models: (i) binary classification and (ii) multiclass classifications. We validate our proposal DNN-IDS by comapring its performance against state-of-the-art machine learning techniques for NIDS.

Structure of the Document

The structure of remainder of this work is as follows.

- **Chapter 1.** deals with the concept of intrusion detection systems. It will also cover the different variants of available IDSs techniques.
- **Chapter 2.** is about deep learning. An introduction to the concept of deep learning along with the different types of machine learning is presented in this chapter.
- **Chapter 3.** covers an overall design of the proposed approach for IDS along with the description of the dataset NSL-KDD.
- **Chapter 4.** describes the tools and the environment used in this work. It also presents experimental results of the implementtion of IDSs.

Chapter 1

Intrusion Detection Systems

1.1 Introduction

With the growth of networking and a large volume of valuable information produced such as personal profiles and credit card, the importance of the security measures grows bigger. Hence, network security has become more important than ever.

Among the mechanisms developed to secure information, Intrusion Detection Systems are an among the most important components for network security. They collect and check packets, looking for unnecessary packets and their behaviors. As soon as unnecessary events and packets are detected, security processes take action for deleting or repairing the system. The main focus is on detecting known and unknown attacks in fast networks in order to reduce attacks by shrinking the time gap between the time of the attack and its detection.

Due to its importance, intrusion detection systems have drwan a lot of interest and have been investigated by many researchers since the founding works in this field dating back to the 1980s [1].

In this chapter we will introduce some basics, concepts, terminology, and definitions in computer security. Subsequently we will look at the different variants of available intrusion detection system (IDS) techniques.

1.2 Generalities on Information Security

1.2.1 Information Security

The term security is easiest to define by breaking it into pieces. An information system consists of the hardware, operating system, and application software that work together to collect, process, and store data for individuals and organizations. Thus information systems security is the collection of activities that protect the information system and the data stored in it. A brief and simple definition of information security is the following:

$$Informationsecurity = confidentiality + integrity + availability + authentication$$

There can be no information security without confidentiality, this ensures that unauthorized entities do not intercept, copy, or replicate information. At the same time, integrity is necessary so that only authorized entities can alter information within a system. Finally, information is not secure without authentication determining whether the end entities is authorized to have access.

1.2.2 Basic Terminology

We start by defining some fundamental terms relating to network security. These terms are the foundation for any discussion of network security and are the elements used to measure the security of a network.

Computer security

Computer security is the use of technology, policies, and education to assure the confidentiality, integrity, and availability of data during its storage, processing, and transmission. To secure data, we pursue three activities: prevention, detection, and recovery [1].

Vulnerabilities:

A vulnerability is an inherent weakness in the design, configuration, or implementation of a network or system that renders it susceptible to a threat. Most vulnerabilities can usually be traced back to one of three sources: *poor design, poor implementation, or poor management.*

Attacks:

An attack is a specific technique used to exploit a vulnerability. For example, a threat could be a denial of service. A vulnerability is in the design of the operating system, and an attack could be a "ping of death". There are two general categories of attacks, passive and active.

Threat:

A threat is anything that can disrupt the operation, functioning, integrity, or availability of a network or system. There are different categories of threats. There are natural threats, occurrences such as floods, earthquakes, and storms. There are also unintentional threats that are the result of accidents and stupidity. Finally,

there are intentional threats that are the result of malicious indent. Each type of threat can be deadly to a network.

Security policy:

Security policy is an action plan that a public or private organization establishes in order to reduce security risks. This plan usually includes specific plans such as a defense policy as well as other indirect policies, purchasing policies, and personnel selection, and also establishes control measures for the security of the organization. On brief security policy is a statement of what is, and what is not allowed.

Countermeasures:

Countermeasures are the techniques or methods used to defend against attacks and to close or compensate for vulnerabilities in networks or systems.

1.2.3 The Security Process

- **Protection** Protection mechanisms are used to enforce a particular policy. The goal is to prevent things that are undesirable from occurring [1].
- **Detection** Detection is simply the process of identifying something is true characteristic if not provide an alert or alarm [1].
- **Response** If, upon examination of an alert provided by our detection system, we find that a policy violation has occurred, we need to respond to the situation [1].

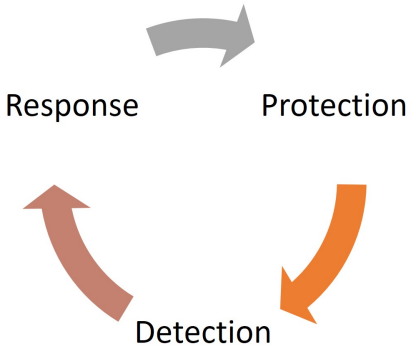

FIGURE 1.1: The security cycle

1.2.4 Information Assurance

The standard model of information assurance is shown in Figure 1.2 [1]. In this model, the security properties of confidentiality, integrity, and availability of information are maintained in the different locations of storage, transport, and processing by technological means, as well as through the process of educating users in the proper policies and practices.
The term assurance is used because we fully expect failures and errors to occur [1].

1.2.4.1 Security Properties

The first aspects of this model we will examine are the security properties that can be maintained. The traditional properties that systems work towards are confidentiality, integrity, and availability, and other properties are sometimes included. Because different applications will have different requirements, a system may be designed to maintain all of these properties or only a chosen subset as needed [1].

1.2.4.2 Information Location

The model of information assurance makes a clear distinction about where information resides within a system. This is because the mechanisms used to protect, detect, and respond differ for each case [1].

1.2.4.3 System Processes

While most computer scientists focus on the technological processes involved in implementing security, technology alone cannot provide a complete security solution. This is because human users are integral in maintaining security. The model of information assurance recognizes this, and gives significant weight to human processes [1].

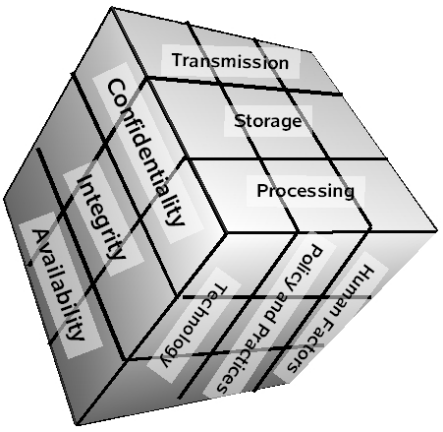

FIGURE 1.2: The standard model of information assurance

1.2.5 Goals of Security

Given a security policy is specification of **secure** and **non secure** actions, these security mechanisms can prevent the attack, detect the attack, or recover from the attack. The strategies may be used together or separately [2].

Prevention means that an attack will fail, prevention involves implementation of mechanisms that users cannot override and that are trusted to be implemented in a correct, unalterable way, so that the attacker cannot defeat the mechanism by changing it.

Detection is most useful when an attack cannot be prevented, but it can also indicate the effectiveness of preventative measures. Detection mechanisms accept that an attack will occur; the goal is to determine that an attack is under way, or has occurred, and report it [2].

Recovery has two forms. The first is to stop an attack and to assess and repair any damage caused by that attack, In a second form of recovery, the system continues to function correctly while an attack is under way [2].

1.3 Intrusion Detection System

To detect abnormal behavior there are many tools and the most frequently mentioned Anti-Virus, Firewall and Intrusion Detection System, IDS is more powerful due to detection capabilities [3].

1.3.1 What is an Intrusion Detection

An intrusion is defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource" [3]. Accordingly, intrusion detection is defined as "the problem of identifying actions that attempts to compromise the integrity, confidentiality, or availability of a computer resource" [3].

1.3.2 Definition of Intrusion Detection System

An IDS intrusion detection system is a piece of software or hardware for monitoring and detecting data traffic or user behavior to identify any type of threat.

1.3.3 History

In 1987, work on IDS was started by Denning et al. [4]. Authors developed a real-time Host Based Intrusion Detection System (HIDS) to detect attack from inside as well as outside the system. For detection of attacks, authors used rule matching mechanism based on audit trail and system log files [5]. Gradually, various researchers contributed to make HIDS more efficient. One more HIDS system which was developed in 1988 is Haystac HIDS [6]. Progressively due to the more usage of computer networks, requirements for detection of attack at network side increased. As a result, researchers started to work on Network Based Intrusion Detection System (NIDS). System presented in [7] is one of the well known NIDS which was developed in 1998.

1.3.4 A Simple Taxonomy of Intrusion Detection Systems

There are various ways to classify the intrusion detection system. Figure 1.3 shows various approaches to classify the IDS. Following are the details about well known and frequently used approaches [8].

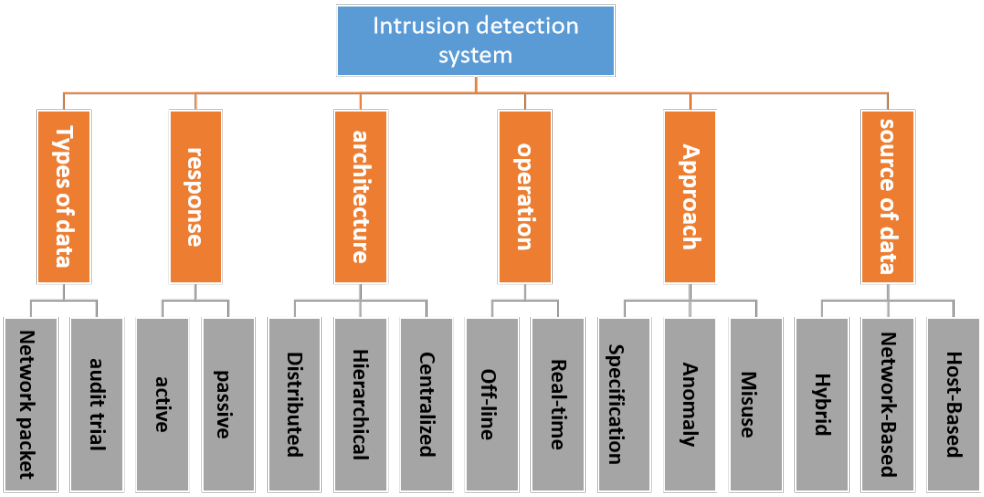

FIGURE 1.3: Types of IDS

1.3.4.1 Host-Based versus Network-Based Intrusion Detection

- **Host-Based System**

  The host-based intrusion detection systems (HIDS) analyzes the data that originates on computers (hosts), such as application and operating system event logs, system all traces. Such systems are effective for insider threats. Abuse of privileges by insiders, accesses of critical data are some of the attacks which can be detected by these systems.

- **Network-Based System**

  The network-based intrusion detection systems (NIDS) process the data that originates on the network, such as TCP/IP traffic. Malformed packets, packet flooding, probes are some of the attacks which can be detected by such systems.

- **Hybrid System**

  The Hybrid mean host data is combined with network information to form a comprehensive view of the network. The main reason for introducing such hybrid IDS is the need to work on-line with encrypted networks and their data destined to the single host (only the source and destination can see decrypted network traffic).

The two general architecture HIDS and NIDS described are two different ways of detecting intrusions. As mentioned intrusion detections can be deployed on different areas, like within a computer to spot users attempting to gain access to which they have no access right, or monitoring network traffic to detect other kind of intrusions like worms, Trojan horses or to take control of a host by yielding an illegal root shell, etc.
As for this project we will concentrate our efforts on one type of intrusion detection, that is Network-Based IDS.

1.3.4.2 Misuse Detection versus Anomaly Intrusion Detection

- **Misuse detection technique**

  Misuse detection or known as **signature-based**, look for well-defined patterns of known attacks where the pattern may be signature, protocol, rule, state, system call and may more. The known attacks are represented as patterns or signatures. Misuse detection is therefore, simply a problem of matching patterns of attack in the given source of data. Such systems detect patterns of known attacks quite accurately and efficiently, and generate very few false alarms.

  The Limitation of Misuse Detection

  - Cannot detect novel, unknown attacks or variations of known attacks.
  - Requires the nature of attacks to be well understood
  - Necessitates that human experts work on the analysis and representation of attacks

- **Anomaly detection technique**

  Anomaly detection is based on the normal behavior of the subject (e.g., a user, program or a system). Any action that significantly deviates from the normal behavior is considered as intrusive. Such systems build a statistical or *machine*

learning model of normal behavior of the subject. The model is basically a list of metrics or patterns that capture the normal profile. The system flags an intrusion if any observed metrics or patterns of given behavior significantly deviate from the model. Such systems detect previously unknown patterns of attacks.

The Limitation of Anomaly Detection

- Generate many false positives (normal behavior classified as intrusive).
- The difficulty of handling gradual misbehavior and expensive computation
- May miss known attacks

- **Specification-based**

  Specification-based detection technique: It is hybrid technique which merges the objective of anomaly and misuse detection techniques. It is mainly focused on identifying deviations from regular behavior.

The above listed two techniques used in intrusion detection system misuse and anomaly detection. Our interest lie in anomaly detection system and later we are going to develop an automated and machine learning approach for NIDS.

1.3.4.3 Real-time versus Off-line Intrusion Detection

- **Real-time**

  A real-time IDS monitors the system continuously and reports intrusions as soon as they are detected. Such systems can substantially reduce the damage to the system, if the system administrator can be notified as early as possible. Moreover, there is a great chance of stopping the attack currently in progress and catching the intruder as intruder would not get much time to delete his trail (e.g., by erasing logs).

- **Off-line**

  An off-line IDS inspects system logs at periodic intervals and then discovers any suspicious activity that was recorded. Such systems are very effective in correlating attacks that span multiple hosts, slow probing attacks that span over hours and days, and for forensic analysis. An off-line IDS typically reduces system overhead but gives much less timely notification of intrusions.

1.3.4.4 Passive versus Active Intrusion Detection

- **Active**

  An active IDS (now called intrusion prevention system IPS) is a system that is configured to automatically block suspected attacks in progress without any intervention required by an operator. IPS has the advantage of providing real-time corrective action in response to an attack but has many disadvantages as well.

- **Passive**

  A passive IDS is a system that is configured only to monitor and analyze network traffic activity and alert an operator to potential vulnerabilities and attacks. It is not capable of performing any protective or corrective functions on its own. The major advantages of passive IDSs are that these systems can be easily and rapidly deployed and are not normally susceptible to attack themselves.

1.3.4.5 Depending on the Architecture:

The most common IDS architecture are:

- **Centralized**

  In centralized IDS, the data may be collected from various sources (hosts or networks) but is sent to a centralized location where it is analyzed. Such systems limit the system scalability as it could become bottleneck on increasing number of sources and also represent a single point of vulnerability.

- **Hierarchical**

  In hierarchical IDS, some of the data collected from multiple hosts or a single host is passed up through the layers and is analyzed to varying degree at each level.

- **Distributed**

  In Distributed IDS, the data is collected and analyzed across the entire network being monitored and results are then sent to a centralized location. Such systems are scalable and not subject to a single point of failure.

1.3.5 Placement of NIDS

Network Intrusion Detection Systems are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. A large NIDS server can be set up on a backbone network, to monitor all traffic, or smaller systems can be set up to monitor traffic for particular server, switch, gateway, or router. It can be shown in Figure 1.4 that IDS could be placed in (Local Area Network) **LAN**, (DeMilitarized Zone) **DMZ** or Internet area.

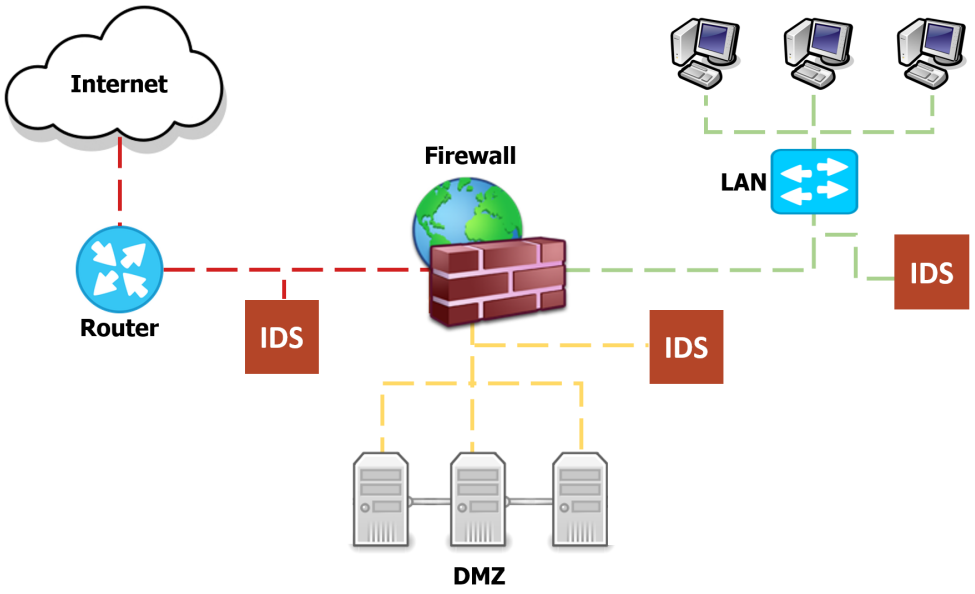

FIGURE 1.4: Computer network with intrusion detection systems

1.3.6 Types of Alarms

An IDS takes input and classifies it as a normal or an attack. This input can be in the form of network traffic, system calls and their sequences, commands and their sequences, user behavior, system behavior and many more. The IDS applies its detection algorithm and classifies the input as a normal or an attack and plays alarms accordingly. Following are the list of various alarms with their meaning [9]:

1. **True Positive:** The input is an attack which is detected by IDS as an attack.
2. **True Negative:** The input is normal which is detected by IDS as normal traffic.
3. **False Positive:** The input is normal which is detected by IDS as an attack.

4. **False Negative:** The input is an attack which is detected by IDS as normal.

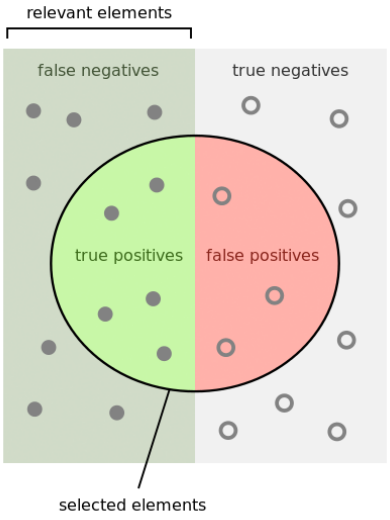

FIGURE 1.5: Alarms

1.3.7 Desirable Characteristics of IDS

As per [10] and [11], following are the desirable characteristics of IDS:

1. Minimum Human Supervision: It must require minimum human supervision.
2. Fault Tolerant: It must continue operating even after system failure.
3. Recoverable: It must be recoverable from the crashes which might be due to hardware or software crashes.
4. Resistant to Attacker: It must periodically check itself to identify whether attacker has modified its own behavior or working.
5. Minimal Overhead: It must require minimum hardware resources so that it does not affect the normal operation of the system.
6. Configurable: It must have facility to configure as per the policy requirement of the department.
7. Updatable: It must be able to update itself by automated process or by updates sent by central authority.
8. Accuracy: It must have high accuracy. Number of false alarm rate should be low

9. Completeness: It should be able to detect all the attacks. This is very difficult to characterize as it is not possible to have knowledge about all the possible attacks.

10. Timeliness: It should be able to give quick response.

1.3.8 Challenges in Intrusion Detection

Intrusion detection will develop towards distributed, intelligent, high detection speed, high accuracy and high security. And the research focus of intrusion detection will include the following.

- **Distributed intrusion detection**

  Distributed intrusion detection system is mainly for large networks and heterogeneous system, which uses distributed structure, collaborative processing and analysis of a variety of information, and a single architecture of intrusion detection system compared with greater detection ability[12].

- **Intelligent intrusion detection**

  Intelligent intrusion detection method is the present stage, including machine learning, neural networks, data mining, and other methods. It has carried out various intelligent techniques in the application and research of intrusion detection. The main purpose of the study is reduced detection system false alarm and false alarm probability, improve the system self learning ability and real-time response. From the current research results, the intrusion detection method based on intelligent technology has many advantages, and has good development potential [13].

- **Intrusion detection based on protocol analysis:**

  The calculation amount of intrusion detection based on protocol analysis is relatively small. It can be used to detect the presence of a high degree of regularity of network protocol, even in high load network, it is not easy to generate packet loss [14].

- **Combined with operating system:**

  Closely integrated with the operating system can enhance the intrusion detection system to new attack detection capabilities.

- **Application layer intrusion detection:**

  The semantics of many intrusions can be understood only in the application layer, and the detection of this kind of intrusion needs to be realized by analyzing the application layer [15].

- **High speed packet capture technology:**

  For network intrusion detection system, high-speed packet capture can reduce the resource consumption and improve the detection speed.

- **Efficient pattern matching algorithm:**

  As intrusions become more diverse and complex, more and more complex models need to be stored in rule base. And complexity of intrusion model definition are higher and higher. Therefore, it is urgent to research and use efficient pattern matching algorithm [16].

- **Test and evaluation of intrusion detection system:**

  The establishment of common intrusion detection system evaluation method and testing platform, which is very important to promote the application and popularization of intrusion detection system, has become another important direction of intrusion detection research [17].

- **Standardization of intrusion detection system:**

  There is no formal international standards of intrusion detection system so far. And it is not conducive to the development and application of intrusion detection system.

- **The interaction between IDS and IDS and other security components:**

  Intrusion detection system could combine with other IDS or security components by cascaded connection or integration.

- **Research on the security of intrusion detection system itself:**

  Intrusion detection system has its own security problem as well. And there should be research on how to protect itself against network attacks.

1.3.9 Conclusion

In this chapter, we have seen an overview about network security. Then we defined the concept of IDS.

Furthermore we mentioned the two main methods of the IDS: the misuse and anomaly detection methods which were described together with their strengths and weaknesses. The differences have been presented and it has been decided to build our an IDS based on anomaly detection technique. That is due to the fact that an anomaly detection technique IDS is more automated and can detect an unknown attack.

Finally we cited the most desirable characteristics and the challenges of intrusion detection system. The anomaly detection NIDS we want to build is another proposal to how to make IDS more automated and with minimum intervention from experts.

Chapter 2

Machine Learning and Deep Learning

2.1 Introduction

To improve the detection efficiency, some artificial intelligence algorithms, have been adopted to solve the intrusion detection problem. In this project we going to to adopt deep learning as a technique used in anomaly intrusion detection.

In this chapter, we provide the theoretical background necessary for understanding the methods discussed in the next chapter. Then, we discuss relevant details of machine learning, neural networks, and deep learning.

2.2 Artificial Intelligence

In the past few years, artificial intelligence (AI) has been a subject of intense media hype. Machine learning, deep learning, and AI come up in countless articles, often outside of technology-minded publications [18].

2.2.1 Artificial Intelligence, Machine Learning, and Deep Learning

Artificial intelligence was born in the 1950s, when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to think a question whose ramifications we are still exploring today. A concise definition of the field would be as follows the effort to automate intellectual tasks normally performed by humans. As such, AI is a general field that encompasses machine learning and deep learning (see Figure 2.1), but that also includes many more approaches that do not involve any learning [18].

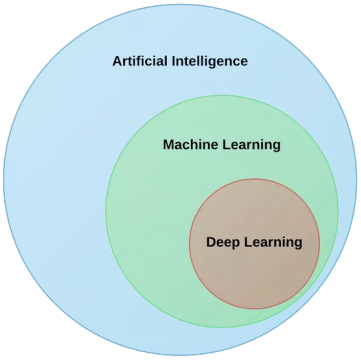

FIGURE 2.1: Artificial intelligence, machine learning, and deep learning

2.3 Machine Learning

Machine learning arises from this question: could a computer go beyond what we know how to order it to perform and learn on its own how to perform a specified task? Could a computer surprise us? Rather than programmers crafting data-processing rules by hand, could a computer automatically learn these rules by looking at data? This question opens the door to a new programming paradigm. With machine learning, humans input data as well as the answers expected from the data, and out come the rules. These rules can then be applied to new data to produce original answers (see Figure 2.2) [18].

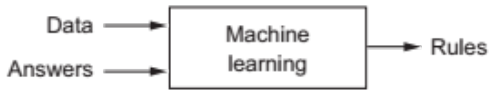

FIGURE 2.2: Machine learning a new programming paradigm

A machine-learning system is trained rather than explicitly programmed. It is presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task [18].

Although machine learning only started to flourish in the 1990s, it has quickly become the most popular and most successful subfield of AI, a trend driven by the availability of faster hardware and larger datasets. Machine learning is tightly related to mathematical statistics, but it differs from statistics in several important ways. Unlike statistics, machine learning tends to deal with large, complex datasets (such as a dataset of millions of images, each consisting of tens of thousands of pixels) for which classical statistical analysis such as Bayesian analysis would be impractical. As a result, machine learning, and especially deep learning, exhibits comparatively little mathematical theory maybe too little and is engineering oriented. It is a hands-on discipline in which ideas are proven empirically more often than theoretically [18].

2.3.1 Types of Machine Learning System

There are so many different types of machine learning systems that it is useful to classify them in broad categories based on:

2.3.1.1 Supervised/Unsupervised Learning

Machine learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and reinforcement learning [19].

Supervised Learning In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels (Figure 2.3) [19].

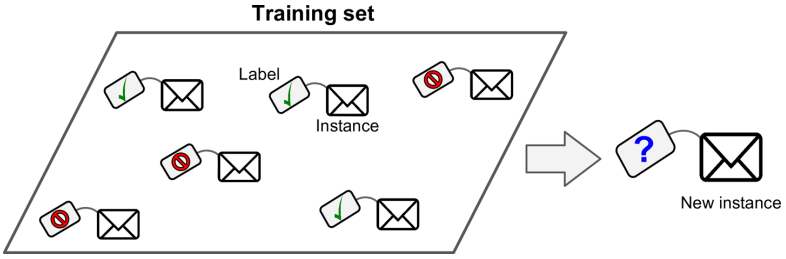

FIGURE 2.3: A labeled training set for supervised learning

Here are some of the most important supervised learning algorithms:

- k-Nearest Neighbors (KNN)
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks (NN)

Unsupervised Learning In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher. (Figure 2.4)[19].

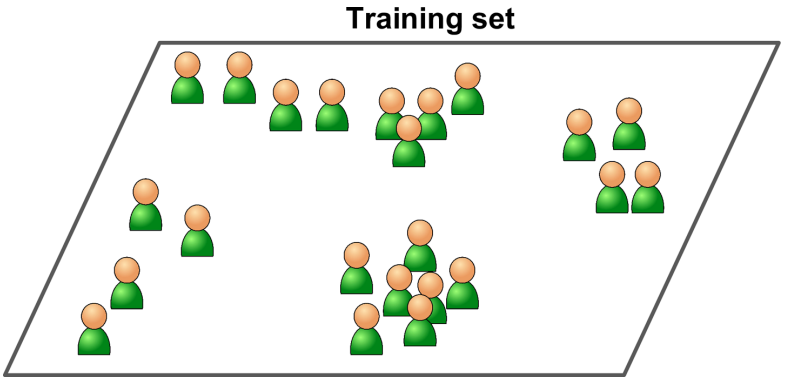

FIGURE 2.4: An unlabeled training set for unsupervised learning

Here are some of the most important unsupervised learning algorithms:

- **Clustering** (e.g., k-Mean, Hierarchical Cluster Analysis HCA)
- **Visualization and dimensionality reduction** (e.g., Principal Component Analysis PCA, Kernel PCA, Locally-Linear Embedding LLE)
- **Association rule learning** (e.g., Apriori, Eclat)

Semisupervised Learning Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning (Figure 2.5) [19].

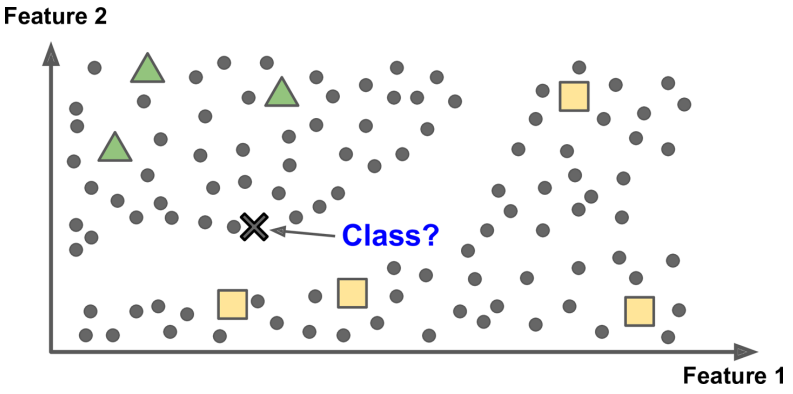

FIGURE 2.5: Semisupervised learning

Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsupervised components called restricted Boltzmann machines (RBMs) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques [19].

Reinforcement Learning Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, as in Figure 2.6). It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation [19].

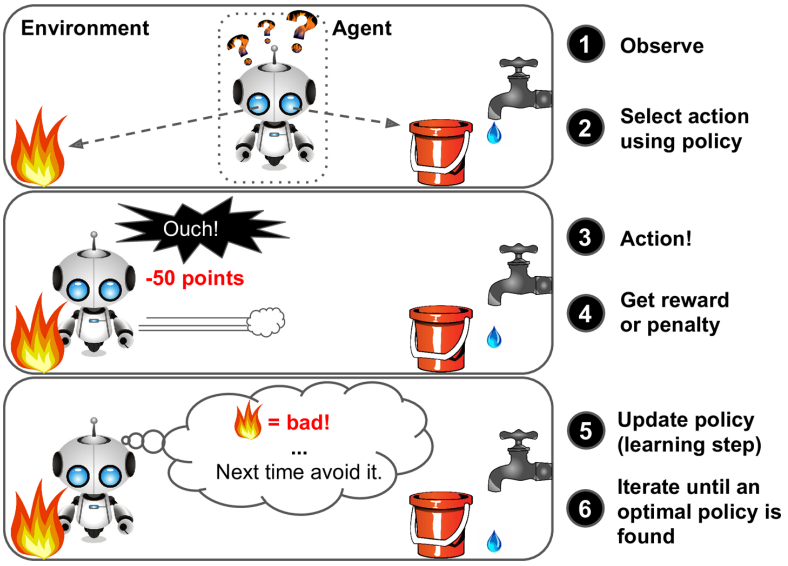

FIGURE 2.6: Reinforcement learning

2.3.1.2 Batch and Online Learning

Another criterion used to classify machine learning systems is whether or not the system can learn incrementally from a stream of incoming data.

Batch Learning In batch learning, the system is incapable of learning incrementally it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning [19].

Online Learning In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches. Each learning step is fast and cheap, so the system can learn about new data on the fly,

as it arrives. Online learning is great for systems that receive data as a continuous flow and need to adapt to change rapidly or autonomously [19].

2.3.1.3 Instance-Based Versus Model-Based Learning

One more way to categorize machine learning systems is by how they generalize. There are two main approaches to generalization: instance-based learning and model-based learning [19].

Instance-Based Learning the system learns the examples by heart, then generalizes to new cases using a similarity measure (Figure 2.7) [19].

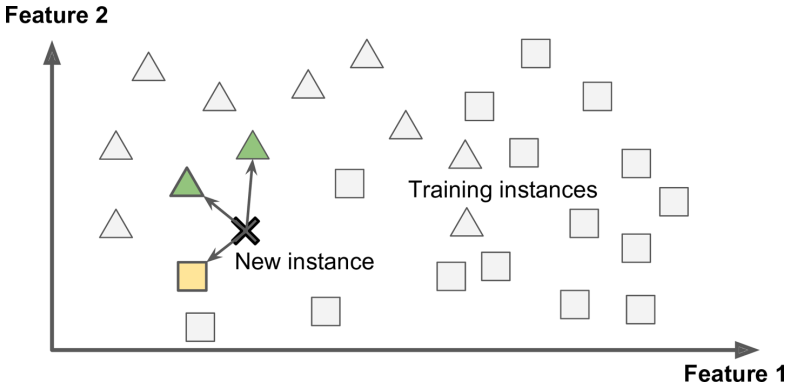

FIGURE 2.7: Instance-based learning

Model-Based Learning Another way to generalize from a set of examples is to build a model of these examples, then use that model to make predictions. This is called model-based learning (Figure 2-8) [19].

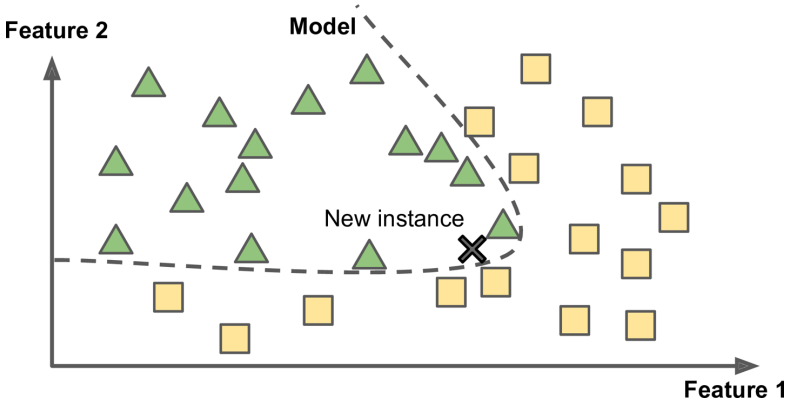

FIGURE 2.8: Model-based learning

2.4 Deep Learning

The performance of machine learning systems strongly depends on the representation of the input data. For many years the research in the field has focused on proper methods for feature extraction, transformation and selection [20]. In standard pattern recognition approaches [21], these features were normally hand-designed and later exploited by classification algorithms.

Differently to past approaches, the idea of representation learning is to jointly discover not only the mapping from input feature to output, but also the features itself [22].

Deep learning [23] follows the philosophy of representation learning and aims to progressively discover complex representations starting from simpler ones. The principle of composition, on the other hand, can be used to describe the world around us efficiently.

The deep learning paradigm is currently implemented with Deep Neural Networks (DNNs), that are Artificial Neural Networks (ANNs) based on several hidden layers between input and output (see Figure 2.9). Each layer learns higher-level features that are later processed by the following layer [24]. When a suitable high-level representation is reached, a classifier can perform the final decision. Modern DNNs provide a very powerful framework for supervised learning: when adding more layers, in fact, a deep network can represent functions of increasing complexity and can potentially reach higher levels of semantic representations.

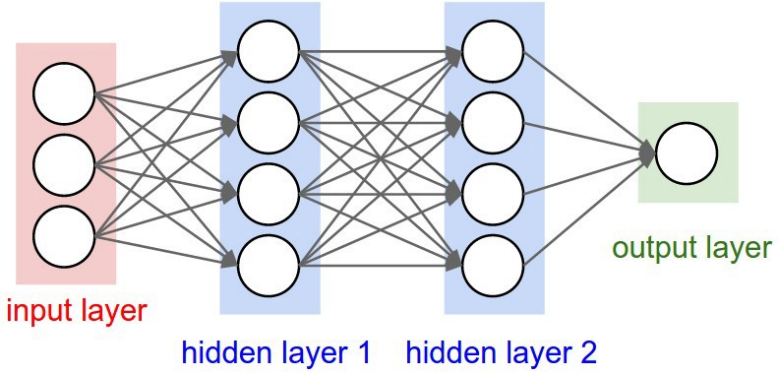

FIGURE 2.9: Neural network architecture

ANNs have been object of several research in the past decades [25]. These efforts were extremely important for the research community, since they laid the foundations for the basic learning algorithms [21]. For instance, the back-propagation algorithm was

invented in the 60s-70s with the contributions of many scientists [26]. Despite these achievements, the time was not yet ripe for the explosion of this technology. The current rise of deep learning can be explained with the following motivations:

- **Big Data:** A key ingredient for the success of this technology is the availability of large datasets. Current systems, in fact, aim to incorporate considerable knowledge into a machine, requiring lots of data. More precisely, the effectiveness of DNNs strongly depends on the capacity of the model, that can be increased by adopting deep and wide architectures. The improved network capacity, however, increases the number of parameters, inherently requiring more data to reliably estimate them. Fortunately, the rapid spread of internet and smartphones, allows easy and cheap big-data collections.

- **Computational power:** To properly exploit deep models and large datasets, a considerable computational power is required. In the last years, important progresses have been done to develop specialized hardware for deep learning. Modern Graphical Processor Units (GPUs), for instance, are currently used by most of deep learning practitioners to efficiently train complex models.

- **Computationally efficient inference:** DNNs usually require a lot of computational power during the training phase. An interesting aspect is that inference can be computed relatively efficiently, allowing, for instance, the development of real-time speech recognizers or lowlatency dialogue systems.

- **Powerful priors:** Last but not least, deep learning incorporates reasonable assumptions about the world. The basic assumption is the compositionality principle previously discussed, that efficiently describes the complex world around us as a progressive composition of different elements. This assumption acts as a prior knowledge used to defeat curse of dimensionality: among all the possible functions that are able to explain a dataset, deep learning restricts this selection to a smaller sub-set that satisfies the compositionality constraint. This naturally entails a regularization effect, that allows training deep architectures.

2.4.1 Basic Algorithms

This part proposes an overview of the main algorithms and techniques used for deep learning. In particular, some general notions about supervised learning are recalled in sub-section 2.1.1. The back-propagation algorithm is discussed in sub-section 2.1.2, while the main optimization techniques are summarized in sub-section 2.1.3. Regularization methods are finally described in sub-section 2.1.4.

2.4.1.1 Supervised Learning

DNNs are often trained in a supervised fashion [21]. This training modality can be formalized as follows let us assume to have a set of N training, where each element is a pair composed of a feature vector x_i and its corresponding label y_i examples like so $\{(x_1, y_1), ..., (x_i, y_i), ..., (x_n, y_n)\}$.The learning algorithm seeks a function $f : X \rightarrow Y$ that maps the input space X into the output space Y. Deep learning is a form of parametric machine learning, whose function f depends on a set of trainable parameters θ. For each particular choice of θ, a different mapping function f is obtained. The number of possible functions that can be represented by the DNN is called *capacity*.

The goal of supervised learning is to find a function f that is able to "explain" well the training samples. More formally, this implies finding proper values of θ able to minimize a certain performance metric:

$$\hat{\theta} = arg_{\theta} min L(Y, f(X, \theta))$$

The function L is called *loss* (or cost) and, intuitively, should assume low values when the parameters θ lead to an output well-matching with the reference labels.

In the context of the Maximum Likelihood Estimation (MLE), the optimization problem can be reformulated in this way:

$$\hat{\theta} = arg_{\theta} max P(Y|X, \theta)$$

where $P(Y|X, \theta)$ is the conditional probability distribution defined at the output of the DNN. To perform a MLE estimation, a popular choice of L is the Negative Log-Likelihood (NLL) or Cross-Entropy (CE). In this case, the MLE optimization can be

rewritten as:

$$\hat{\theta} = arg_{\theta} min - log(P(Y|X, \theta))$$

Another popular choice is the Mean Squared Error (MSE):

$$\hat{\theta} = arg_{\theta} min \, \|Y - f(X, \theta)\|^2$$

Solving the training optimization problem is challenging, especially for DNNs composed of a huge number of parameters. The functions f originated by DNNs are, in fact, typically non-linear, making the optimization space highly non-convex. Recently, some theoretical and experimental studies have shown that the main challenge are *saddle points* and not local minima as commonly believed in the past [27]. Even though these results are still object of an open debate in the research community, they suggest that the parameter space is flat almost everywhere and the (very rare) local minima are almost all also global ones [28].

There are in principle various ways to solve this optimization problem [29]. A naive solution would be to try all the possible θ and choose the configuration that minimizes the cost function. This approach is clearly unfeasible, especially for a large number of parameters. Another solution would be to exploit evolutionary optimizations, based on genetic algorithms or particle swarm techniques [30]. Despite the interesting aspects of these optimizers (e.g, high parallelism, differentiability is not strictly required), such methods require very frequent evaluations of the cost function, that is impractical for DNNs trained on large datasets. At the time of writing, the most popular choice is gradient-based optimization.

2.4.1.2 Back-Propagation Algorithm

The gradient $\frac{\partial L}{\partial \theta}$ is a very precious information that describes what happens to the cost function L when a little perturbation is applied to the parameters θ. If this perturbation causes an improvement of the loss, it could be convenient to do a little step in the direction indicated by the gradient.

The computation of $\frac{\partial L}{\partial \theta}$ is normally performed with the back-propagation algorithm [31], that is often misunderstood as meaning the whole learning procedure. Actually, back-propagation is only a method for computing the gradient.

Deriving an analytical expression of $\frac{\partial L}{\partial \theta}$ is rather straightforward for systems that are differentiable almost everywhere. In most of the cases, in fact, the analytical expression of the gradient is a direct application of basic calculus rules. As shown in Figure 2.10, a DNN can be described as a composite function that performs a chain of computations. Gradients can thus be computed with the chain rule, as reported in the following

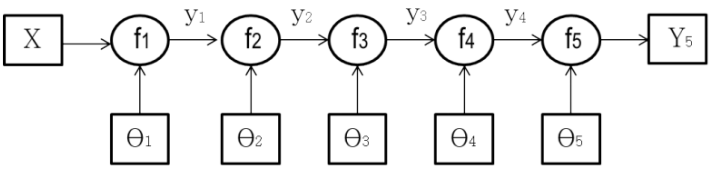

FIGURE 2.10: A composite function representing the computations performed in the various layers of a DNN

Equations:

$$\frac{\partial y_5}{\partial \theta_1} = \frac{\partial y_1}{\partial \theta_1} \cdot \frac{\partial \mathbf{y_2}}{\partial \mathbf{y_1}} \cdot \frac{\partial \mathbf{y_3}}{\partial \mathbf{y_2}} \cdot \frac{\partial \mathbf{y_4}}{\partial \mathbf{y_3}} \frac{\partial \mathbf{y_5}}{\partial \mathbf{y_4}} \tag{2.1}$$

$$\frac{\partial y_5}{\partial \theta_2} = \frac{\partial y_2}{\partial \theta_2} \cdot \frac{\partial \mathbf{y_3}}{\partial \mathbf{y_2}} \cdot \frac{\partial \mathbf{y_4}}{\partial \mathbf{y_3}} \cdot \frac{\partial \mathbf{y_5}}{\partial \mathbf{y_4}} \tag{2.2}$$

$$\frac{\partial y_5}{\partial \theta_3} = \frac{\partial y_3}{\partial \theta_3} \cdot \frac{\partial \mathbf{y_4}}{\partial \mathbf{y_3}} \cdot \frac{\partial \mathbf{y_5}}{\partial \mathbf{y_4}} \tag{2.3}$$

$$\frac{\partial y_5}{\partial \theta_4} = \frac{\partial y_4}{\partial \theta_4} \cdot \frac{\partial \mathbf{y_5}}{\partial \mathbf{y_4}} \tag{2.4}$$

Beyond the mathematical expression, the intuition behind the chain rule is this: the little change applied to the parameter θ_1 causes a change on the output of function f_1. This perturbation will propagate until the end of the chain, eventually causing a perturbation on the final node, that typically computes the cost function L.

Despite the relative simplicity of the gradient expression, its numerical evaluation can be computationally expensive. A naive (but inefficient) solution would be to independently evaluate the gradient expression for each parameter. The back-propagation algorithm [31] is a simple and inexpensive procedure that performs the various operations in a specific order. From the previous equations, one can notice that there are several shared

computations. It would be, for instance, very convenient to start from the last equation, store the results in bold, and reuse them to compute the following gradients.

The back-propagation algorithm is based on a dynamic programming approach and is summarized by the following step:

1. **Forward Propagation:** propagate the input features to the output and store the activations y.
2. **Compute Loss:** evaluate the loss function L at end of the chain.
3. **Back Propagation:** compute the gradient from the last element of the chain to the first one.

When propagating gradients through long computational chains, however, some issues might arise [32]. The chain rule, in fact, implies several gradient multiplications that can lead to *vanishing* or, less often, to *exploding gradients* [33].

Exploding gradients can effectively be tackled with simple clipping strategies [32], while vanishing gradient is more critical and might impair training very deep neural networks.

2.4.1.3 Optimization

Once computed, the gradient has to be exploited by an optimizer to progressively derive better parameters.

The most popular optimization algorithm is Gradient Descend (GD), that updates θ according to the following equation:

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta} \tag{2.5}$$

The parameters are updated in the direction pointed by the gradient, with a step size determined by the learning rate η (the minus is due to the minimization of the loss see Figure 2.11). Note that gradient-based optimization does not provide any guarantee on global optimality. In such complex high dimensional spaces, a crucial role is thus played by a proper initialization of the θ and by a suitable choice of loss and activation functions.

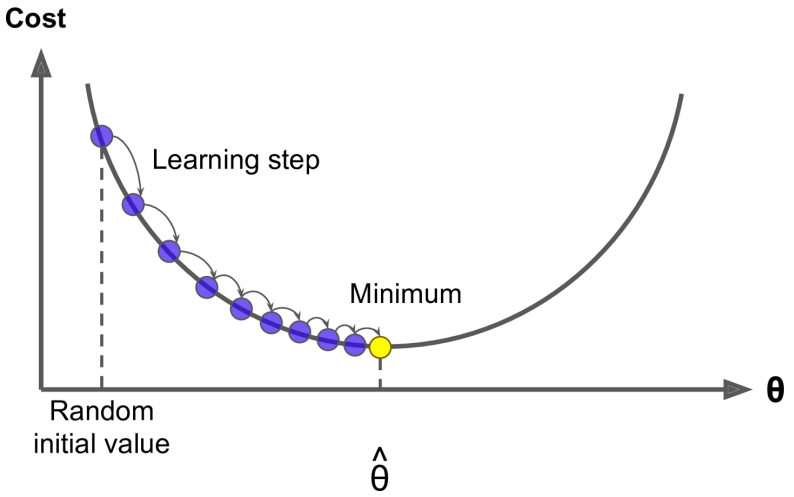

FIGURE 2.11: Gradient descent

Depending on how many data are used to compute the gradient, some variants of GD can be defined. When the gradient is computed on the full training dataset, the optimizer is called batch-GD. A popular alternative is Stochastic Gradient Descend (SGD) that splits the dataset into several smaller chunks (called mini-batches) and update the parameters more frequently, with well-known benefits in terms of both accuracy and training convergence. Below is a graph that shows the gradient descent's variants and their direction towards the minimum:

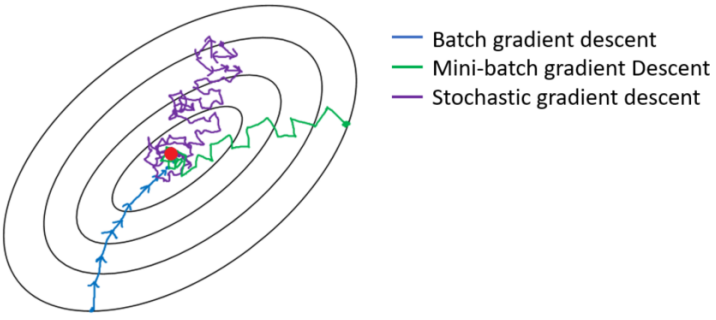

FIGURE 2.12: Gradient descent variants trajectory towards minimum

Standard SGD, however, has trouble navigating on areas where the surface curves much more steeply in one dimension rather than in another. To mitigate this issue, a *momentum* is often applied to the update equations for accelerating the training convergence [34].

Another issue is that the same learning rate applies to all the parameters. This stiffness can be critical, since each θ has its own characteristics, possibly requiring independent learning rates. For instance, if the absolute value of the gradient is very large in a specific dimension, it means that we are in a very steep area and would be more prudent to do a little step using a small learning rate. On the contrary, when the gradient is small, a higher learning rate can be used. Following this philosophy, several variations of SGD, such as Adagrad [35], Adadelta [36], RMSprop, Adam [37], have been recently proposed. These solutions, in general, rescale η by gradient-history metrics, resulting in a faster and more robust optimization.

2.4.1.4 Regularization

crucial challenge in machine learning is the ability to perform well on previously unobserved inputs. This ability is called *generalization* [23]. Possible causes of poor generalization are *underfitting* and *overfitting*, that are two central issues when training DNNs. **Underfitting** occurs when the model is not able to reach a sufficiently low error on the training set. This might arise when the DNN has not enough capacity. **Overfitting** occurs when the gap between training and test errors is too large. Differently to underfitting, this might happen when the model has excessive capacity. The strategies to counteract overfitting are known as *regularization*, and normally consist of methods for reducing the network capacity based on prior knowledge. From this point of view, even deep learning itself can be regarded as a regularization technique, due to the prior knowledge naturally embedded in the compositionality principle.

The great importance of these methodologies has made regularization one of the major research directions in the field. In the following, the most popular regularizers are described.

L2 Regularization Many regularization approaches are based on limiting the model capacity by adding a parameter norm penalty to the loss function. These approaches tend to penalize too complex solutions, following the philosophy of the **Occam's razor** principle: *"among competing hypotheses leading to the same performance, the simpler one should be selected"*. The most common method following this approach is the L^2

regularization, that add a penalty to the loss function L:

$$\tilde{L} = L + \alpha \|\theta\|^2 \tag{2.6}$$

where α is an hyperparameter that weights the contribution of the regularization term. The insight behind this kind of regularization is that solutions characterized by higher norms of the the trainable parameters are more complex and are more likely to overfit the training dataset.

Dropout An effective way to improve generalization is to combine several different models [38]. If each classifier has been trained separately, it might have learned different aspects of the data distribution and their mistakes are likely to be complementary. Combining them helps produce a stronger model, that is hopefully less prone to overfitting. Even though these methods are very effective, a major limitation lies in the considerable computational efforts needed to train and test different DNNs. Dropout [39] is an effective regularization method that provides an inexpensive approximation to training and evaluating an exponential number of neural networks.

The key idea is to randomly drop neurons during training with a probability called dropout rate ρ. This way, a subnetwork is sampled from an exponential number of smaller DNNs for each training sample. At test time, the whole network is used (i.e., the DNN with all the neurons active), but the activations are scaled down by ρ.

This ensemble learning approach significantly reduces overfitting and gives major performance improvements. Many variants of dropout have been proposed in the literature [40, 41]. For instance, in [42] the regularizer is applied to the weight connections rather than on neurons, while in [43, 44] dropout is extended to recurrent neural networks.

Data Augmentation The best way to achieve generalization would be to train the model with more data. However, in practice, the amount of data is limited and the collection of large annotated corpora is very expensive. A possible alternative is to artificially process the available data, in order to generate novel training samples. For a computer vision application, one can for instance, rescale, rotate, or shift the available images to generate novel samples. For speech recognition, one way is to apply proper algorithms able to modify pitch, formants, and other speaker characteristics. Data

augmentation can also be regarded as a way for adding prior knowledge to a model, since we exploit the information that the new samples do not change the class label when are processed.

Other Regularizers Other approaches have been proposed for counteracting overfitting. A popular way is to add random noise during learning. Several methods have been proposed in the literature, proposing to add it at gradient, weight, input, output or hidden activation levels [45–47].

An alternative consists in adopting a semi-supervised approach, where unsupervised data can be exploited as prior knowledge to improve generalization. From this point of view, the pre-training approach based on Restricted Boltzman Machines (RBM) [48] or autoencoders [49] can be regarded as a form of regularization. Multi-task learning (i.e., building DNN solving multiple correlated tasks) can also be considered as a sort of regularization, since it encourages the DNN to discover very general features at the first hidden layers.

Finally, one the most popular and simple approach for regularization is early stopping [50]. The idea is to periodically monitoring the performance of the DNN on held-out data and stop the training algorithm when this performance begins to deteriorate.

2.4.1.5 Hyperparameter Selection

Most deep learning algorithms are based on some hyperparameters that must be properly set to ensure a good performance. The most important ones describe the structure of the network (e.g., number of hidden layers, number of hidden units per layer), determine the optimization characteristics (e.g., learning rate), and specify the behaviour of the regularizer (e.g., weight decay or dropout rate). Properly setting the hyperpameters is rather difficult, mainly because a new model should be trained for each new setting. Moreover, several hyperparameters can be correlated each other, making an independent optimization of them usually not viable.

There are two basic approaches to derive them: choosing them manually and choosing them automatically. The manual selection requires a considerable experience and

familiarity with the addressed task as well as a precise knowledge of the role of each specific hyperparameter. This is possible for well-explored machine learning tasks, where a detailed literature suggesting reasonable settings is available.

When the manual approach is not feasible, a possible alternative is to automatically select the hyperparameters with grid search. Grid search simply tries all the combinations over a specified range of values. Although this search can be easily parallelized, its computational expense is exponential in the number of hyper-parameters. A straightforward alternative is to sample them randomly [51]. This approach is still very easy to parallelize, and is generally faster than grid search.

2.4.2 Main Architectures

This section summarizes the main architectures used in deep learning.

2.4.2.1 Neuron Activations

The computations performed by the DNNs are a sequence of linear operations followed by non-linear ones. More precisely, the output of a hidden layer h_{i+1} composed of n neurons and fed by m input features h_i, can be represented as follows:

$$h_{i+1} = g(\underbrace{Wh_i + b}_{a}) \tag{2.7}$$

Where W is the weight matrix $(n * m)$, b is the bias vector of n element and g is the activation function. The linear transformation performed before applying g is called affine transformation a. The choice of g is particularly important, and several research efforts have been devoted to study proper activation functions. The most popular are:

- **Linear:** The simplest neurons are obtained by directly taking the affine transformation without any non-linearity. These neurons are called linear and can be used, for instance, for predicting real numbers in the output layer (regression problem).

- **Sigmoid;** For several decades, the most popular activation was the logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + exp(-x)} = \frac{exp(x)}{1 + exp(-x)} \tag{2.8}$$

The main advantage is that such activations are bounded between 0 and 1. However, the use of sigmoid activations in modern feedforward networks is now discouraged, since they are characterized by close-to zero gradients across most of their domain. In fact, $\sigma(x)$ saturates when their input is very positive or very negative, slowing down the training.

- **Hyperbolic Tangent:** This kind of non-linearity is a rescaled version of the sigmoid function:

$$tanh(x) = \frac{exp(x) - exp(-x)}{exp(x) - exp(+x)} \qquad = 2\sigma(2x) - 1 \tag{2.9}$$

It suffers from the main disadvantages of sigmoid units, but in general it provides better performance. This is due to the fact that tanh is symmetric around zero, making this activation less prone to saturation in the last layers.

- **Rectified Linear Units:** This kind of activation is currently the most popular choice in modern feed-forward neural networks [52, 53]:

$$ReLU(x) = max(0, x) \tag{2.10}$$

The main insight behind the use of these activations is that they are rather similar to linear units. Linear units, as we have outlined before, lead to a convex optimization space, that is very easy to optimize. ReLUs inherit, at least in part, the benefits of the latter activations. More precisely, the derivatives of a $ReLU$ remain large whenever the unit is active.

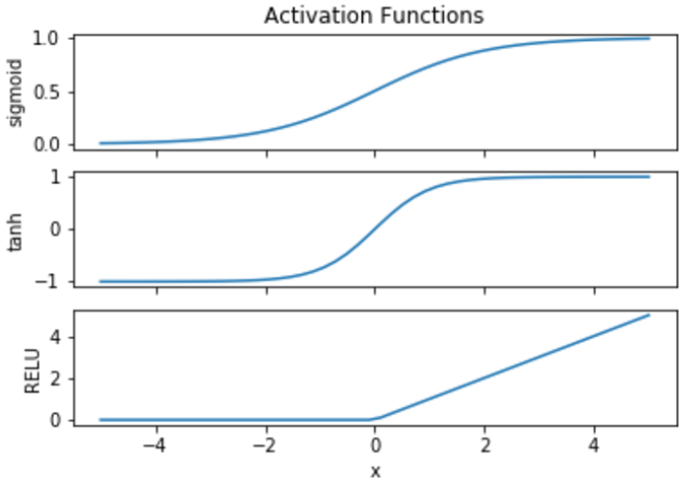

FIGURE 2.13: Main activation functions used in deep neural networks.

- **Softmax:** Softmax neurons are often used in the output layer to estimate a probability distribution over a set of n alternatives. To represent probabilities, each output neuron must assume values between 0 and 1, and the sum of all of them must be 1. Formally, the softmax of the i^{th} neuron is defined as follows:

$$softmax(a_i) = \frac{exp(a_i)}{\sum_{j=1}^{n} exp(a_j)} \tag{2.11}$$

2.4.2.2 Feed-forward Neural Network

The prediction $P(y = x)$ in Feed-Forward DNNs (FF-DNN) is just a function of the current input x and of the parameters θ:

$$y = f(x, \theta) \tag{2.12}$$

These models are called feed-forward because the information flows from the input to the output without any feedback connection. An example of feed-forward neural network composed of four hidden layers is shown in Figure 2.14. The figure depicts a popular architecture called fully-connected

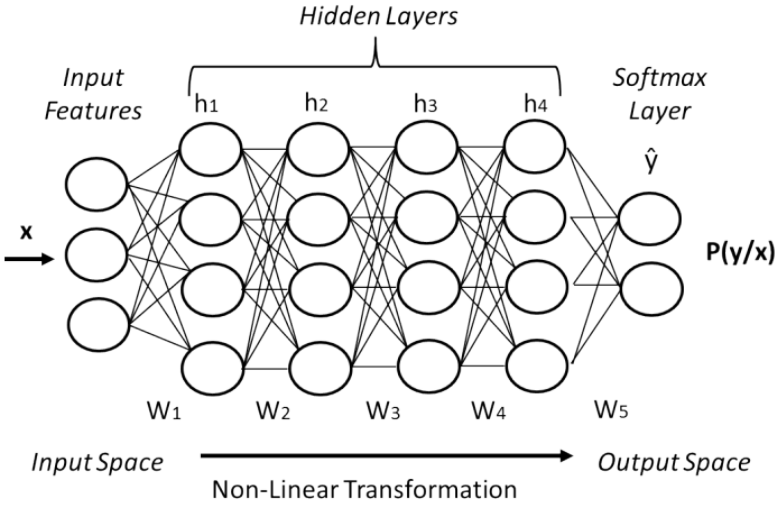

FIGURE 2.14: A fully-connected composed of four hidden layers

DNN or fully-connected, where all the neurons are connected with the ones of the following layer.

An alternative to fully-connected neural neural networks are Convolutional Neural Networks (CNNs) [54]. Differently to the former ones, CNNs are based on local connectivity, weight sharing and max pooling. The combination of these characteristics make CNNs particularly suitable for managing correlations across features. Moreover, the presence of max pooling allows the network to obtain shift-invariant properties. CNNs are inspired by biological studies of the visual cortex and are extremely successful in practical applications, especially in computer vision [55].

2.4.2.3 Batch normalization

Training DNNs is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This problem, known as *internal covariate shift*, slows down the training of deep neural networks.

Batch normalization [56], that has been recently proposed in the deep learning community, addresses this issue by normalizing the mean and the variance of each layer's pre-activation for each training mini-batch. It has been long known that the network training converges faster if its inputs are properly normalized [57] and, in such a way, batch normalization extends this normalization to all the layers of the architecture.

Batch normalization resulted particularly helpful to achieve regularization, to significantly speed-up the convergence of the training phase as well as to improve the overall accuracy of a DNN. The regularization effect is due to the fact that mean and variance normalizations are performed on each mini-batch rather than on the entire dataset.

2.4.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are architectures suitable for processing sequences [58]. The elements of a sequence are, in most of the cases, not independent. This means that, in general, the emission of a particular output might depend on the surrounding elements or even on the full-history. To properly model the sequence evolution, the presence of memory to keep track of past or future elements is thus of fundamental importance. The memory can be implemented using feedback connections, that introduce the concept of state (see Figure 2.15).

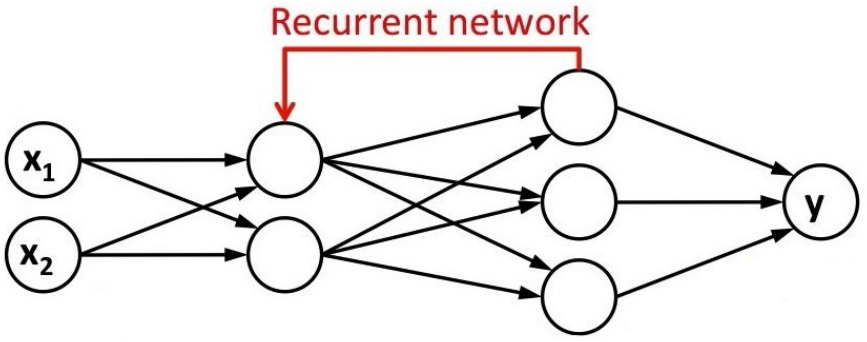

FIGURE 2.15: Recurrent neural networks

2.4.2.5 Architectural Variations

During the last years, several novel architectures have been proposed in the deep learning field this part summarizes the most popular ones.

First of all, architectures based on combinations of convolutional, recurrent and fully-connected layers gained a lot of popularity, especially in the field of speech recognition [59]. Convolutional layers are used for feature extraction, recurrent layers for temporal processing, and fully-connected layers for the final classification.

Other recently-proposed DNNs extend the shortcut idea to feed-forward DNN for improving the propagation of the gradient across the various hidden layers. With this regard, a popular architecture is called Residual Neural Networks (ResNet) [60]. ResNets consider a direct connection through the various hidden layers, that allow the gradient to flow unchanged. By stacking these layers, the gradient can theoretically pass over all the intermediate layers and reach the bottom one without being diminished.

A related idea is exploited in the context of Highway connections [61, 62]. Highway Networks preserve the shortcuts introduced in ResNets, but augments them with a learnable gate that manages the information flow through the hidden layers. The latter networks actually extend the idea of multiplicative learning gates to feed-forward DNNs.

2.4.2.6 Future Challenges

The rapid rise of deep leaning contributed to spread optimism towards this technology. However, despite of such a great enthusiasm, there are still major scientific challenges to address for really reaching higher levels of artificial intelligence [63].

A noteworthy challenge is the effective use of unsupervised data. The recent progress mostly involved supervised learning, that have contributed to achieve state-of-the-art performance in numerous fields. The goal of unsupervised learning is to understand the world around us by observation, which is an ability largely missing in current supervised DNNs.

Another challenge concerns the learning efficiency. Current solutions, in fact, require much more information than humans to learn. For instance, humans are able to learn a simple concept (such as a cat) with few examples, while a machines can model it only with hundreds or even thousands of samples. In the future, the development of learning algorithms able to better disentangle and model the factors of variability will be of primary interest.

Moreover, current deep neural networks are able to solve rather efficiently only single tasks that are generally rather limited and specific (like recognizing faces, classifying sounds, playing Atari). Differently to human brain, current technology is not able to simultaneously solve very different problems at the same time. It would be thus of great

interest the development of more effective multi-task strategies capable of better mimic the human brain.

Another challenge for the future is long-life learning. Current systems are first trained and later tested. The idea of long-life learning is to build a never-ending learning system, that continues to learn from the experience and progressively improves its performance.

Beyond the other challenges, a major achievement would also be the development of a kind of "theory of intelligence" that can better steer the development of intelligent machines. The research in the field, in fact, is now solely based on empirical attempts, that are mostly guided by human intuitions. We can compare the current situation of AI with the first attempts done by the Wright brothers to build a "flying machine". Their approach was only based on a trial-and-error strategy, without any notion about the physics of aerodynamics, whose knowledge is clearly very helpful to design modern aircrafts.

2.5 Related Work

This section presents various recent accomplishments in this area. It should be noted that we discuss only the work which have used the NSL-KDD dataset for their performance benchmarking, therefore, any dataset referred from this point forward should be considered as NSL-KDD. This allows a more accurate comparison of our work with other found in literature.

Many machine learning techniques were used for developing IDS. A Studies on each technique along with their results was discussed clearly in [64]. A number of approaches based on traditional machine learning, including J48, Naive Bayesian, NB Tree, Random Forest, Multi-layer Perceptron, Random Tree and Support Vector Machine, have been proposed and have achieved success for an intrusion detection system.

In [65], the author propose a deep learning approach for intrusion detection using recurrent neural networks (RNN-IDS). and they study the the performance of the model in binary classification and multiclass classification. The experimental results show that RNN-IDS is very suitable for modeling a classification model.

Chapter 3

Intrusion detection using deep learning

3.1 Introduction

After the theoretical study in the two previous chapter. We will describe the NSL-KDD dataset, which is widely used as one of the few publicly available data sets for network-based anomaly detection systems. Therefore we'll describe the global design of the proposed approaches.

3.2 The NSL-KDD Data Set

During the last decade the analysis of intrusion detection has become very important, the researcher focuses on various dataset to improve system accuracy and to reduce false positive rate based on DAPRA 98 and later the updated version as KDD cup 99 dataset which shows some statistical issues, it degrades the evaluation of anomaly detection that affects the performance of the security analysis which leads to the replacement of KDD dataset to NSL-KDD dataset.

3.2.1 Brief KDD CUP 99 Data-Set Description

KDD'99 data set is prepared by Stolfo et al [66]. and is built based on the data captured in DARPA'98 IDS evaluation program . DARPA'98 is about 4 gigabytes of compressed raw (binary) tcpdump data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes. The two weeks of test data have around 2 million connection records. KDD training dataset consists of approximately 4,900,000 single connection vectors each of which contains 41 features and is labeled as either normal or an attack, with exactly one specific attack type.

3.2.2 Dataset Description

The inherent drawbacks in the KDD cup 99 dataset [66] has been revealed by various statistical analyses has affected the detection accuracy of many IDS modelled by researchers. NSL-KDD data set [67] is a refined version of its predecessor.

It contains essential records of the complete KDD data set. There are a collection of downloadable files at the disposal for the researchers. They are listed in the Table 3.1

No	**Name of the file**	**Description**
1	KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
2	KDDTrain+_20Percent.TXT	A 20% subset of the KDDTrain+.txt file
3	KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
4	KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

TABLE 3.1: List of NSL-KDD dataset files and their description

1. Redundant records are removed to enable the classifiers to produce an un-biased result.

2. Sufficient number of records is available in the train and test data sets, which is reasonably rational and enables to execute experiments on the complete set.

3. The number of selected records from each difficult level group is inversely proportional to the percentage of records in the original KDD data set.

3.2.3 Advantages of NSL-KDD

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 difficulty level 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.

3.2.4 Features Details

In each record there are 41 attributes unfolding different features of the flow and a label assigned to each either as an attack type or as normal.

The details of the features namely the features name, their description and sample data are listed in the Tables below. The Table 3.6 contains type information of all the 41 attributes available in the NSL-KDD data set. The 42nd feature contains data about the various 5 classes of network connection vectors and they are categorized as one normal class and four attack class. The 4 attack classes are further grouped as DoS, Probe, R2L and U2R. The description of the attack classes. In NSL-KDD features can be classified into three groups:

1. **Basic features:** this category encapsulates all the attributes that can be extracted from a TCP/IP connection. Most of these features leading to an implicit delay in detection.

2. **Traffic features:** this category includes features that are computed with respect to a window interval and is divided into two groups:

   - **same host features:** examine only the connections in the past 2 seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.
   - **same service features:** examine only the connections in the past 2 seconds that have the same service as the current connection.

   The two aforementioned types of traffic features are called time-based. However, there are several slow probing attacks that scan the hosts (or ports) using a much larger time interval than 2 seconds, for example, one in every minute. As a result, these attacks do not produce intrusion patterns with a time window of 2 seconds. To solve this problem, the same host and same service features are re-calculated

but based on the connection window of 100 connections rather than a time window of 2 seconds. These features are called connection-based traffic features.

3. **Content features:** unlike most of the DoS and Probing attacks, the R2L and U2R attacks don't have any intrusion frequent sequential patterns. This is because the DoS and Probing attacks involve many connections to some host(s) in a very short period of time; however the R2L and U2R attacks are embedded in the data portions of the packets, and normally involves only a single connection. To detect these kinds of attacks, we need some features to be able to look for suspicious behavior in the data portion, e.g., number of failed login attempts. These features are called content features.

No	feature name	description	Sample Data	type
1	**duration**	Length of time duration of the connection	0	continuous
2	**protocol_type**	Protocol used in the connection	tcp	discrete
3	**service**	Destination network service used	ftp_data	discrete
4	**src_bytes**	Number of data bytes transferred from source to destination in single connection	491	continuous
5	**dst_bytes**	Number of data bytes transferred from destination to source in single connection	0	continuous
6	**flag**	Status of the connection Normal or Error	SF	discrete
7	**land**	if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0	0	discrete
8	**wrong_fragment**	Total number of wrong fragments in this connection	0	continuous
9	**urgent**	The number of urgent packets in this connection.Urgent packets are packets with the urgent bit activated	0	continuous

TABLE 3.2: Basic features of individual TCP connections

No	feature name	description	Sample Data	type
10	**Hot**	Number of hot indicators in the content such as: entering a system directory, creating programs and executing programs	0	continuous
11	**Num_failed_logins**	Count of failed login attempts	0	discrete
12	**Logged_in**	Login Status :1 if successfully logged in; 0 otherwise	0	discrete
13	**Num_compromised**	Number of"compromised" conditions	0	continuous
14	**Root_shell**	1 if root shell is obtained; 0 otherwise	0	discrete
15	**Su_attempted**	1 if "su root" command attemptedor used ; 0 otherwise	0	discrete
16	**Num_root**	Number of"root" accesses or number of operations performed as a root in the connection	0	discrete
17	**Num_file_creations**	Number of file creation operations in the connection	0	continuous
18	**Num_shells**	Number of shell prompts	0	continuous
19	**Num_access_files**	Number of operations on access control files	0	continuous
20	**Num_outbound_cmds**	Number of outbound commands in an ftp session	0	continuous
21	**Is_hot_login**	1 if the login belongs to the"hot" list i.e.,root or admin;else 0	0	discrete
22	**Is_guest_login**	1 if the login is a "guest"login; 0 otherwise	0	discrete

TABLE 3.3: Content features within a connection suggested by domain knowledge

No.	feature name	description	Sample Data	type
23	**Count**	Number of connections to the same destination host as the current conn ection in the past two seconds	2	continuous
24	**Srv_count**	Number of connections to the same service (port number) as the current connection in the past two seconds	2	discrete
25	**Serror_rate**	The percentage of connections that have activated the flag (4) s0, s1,s2 or s3, among the connection saggre gated in count (23)	0	discrete
26	**Srv_serror_rate**	The percentage of connections that have activated the flag (4) s0, s1,s2 or s3, among the connection saggre gated insrv_count (24)	0	continuous
27	**Rerror_rate**	The percentage of connections that have activated the flag (4) REJ, am ong the connection saggregated in count (23)	0	discrete
28	**Srv_rerror_rate**	The percentage of connections that have activated the flag (4) REJ, am ong the connection saggregated insrv_count (24)	0	discrete
29	**Same_srv_rate**	The percentage of connections that were to the same service, among the connection saggregated in count (23)	1	continuous
30	**Diff_srv_rate**	The percentage of connections that were to different services, among the connection saggregated in count (23)	0	continuous
31	**Srv_diff_host_rate**	The percentage of connections that were to different destination mach ines among the connection saggreg ated in srv_count (24)	0	continuous

TABLE 3.4: Traffic features computed using a two-second time window

No	feature name	description	Sample Data	type
32	**Dst_host_count**	Number of connections having the same destination host IP address	150	continuous
33	**Dst_host_srv _count**	Number of connections having the same port number	25	continuous
34	**Dst_host_same_ srv_rate**	The percentage of connections that were to the same service, among the connection saggreg ated in dst_host_count(32)	0.17	continuous
35	**Dst_host_diff_ srv_rate**	The percentage of connections that were to different services, among the connection saggreg ated in dst_host_count(32)	0.03	continuous
36	**Dst_host_same _src_port_rate**	The percentage of connections that were to the same source port, among the connection saggregated in dst_host_srv _count (33)	0.17	continuous
37	**Dst_host_srv_ diff_host_rate**	The percentage of connections that were to different destination machines,among the connection saggregated in dst_host_srv_c	0	continuous
38	**Dst_host_ serror_rate**	The percentage of connections that haveactivated theflag (4) s0, s1,s2 or s3, among theconnection saggregated in dst_host_count(32)	0	continuous
39	**Dst_host_srv_ serror_rate**	The percent of connections that haveactivated theflag (4) s0, s1,s2 or s3, among the connection saggregated in dst_host_srv_count (33)	0	continuous
40	**Dst_host_ rerror_rate**	The percentage of connections that have activated the flag (4) REJ, among the connection saggregated in dst_host_count(32)	0.05	continuous
41	**Dst_host_srv _rerror_rate**	The percentage of connections that have activated the flag (4) REJ, among the connection saggregated in dst_host_srv_count (33)	0	continuous

TABLE 3.5: Host based traffic features in a network connection

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4).
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21), is_guest_login(22).
umeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23), srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29), diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41).

TABLE 3.6: Features value type

3.2.5 Attacks Details

The attack classes present in the NSL-KDD data set are grouped into four categories [64, 66] :

1. **Denial of Service Attack (DoS):** Denial of service is an attack category, which depletes the victim's resources thereby making it unable to handle legitimate requests e.g. syn flooding. Relevant features: source bytes and percentage of packets with errors.

2. **Probing Attack:** Surveillance and other probing attack's objective is to gain information about the remote victim e.g. port scanning. Relevant features: duration of connection and source bytes.

3. **User to Root Attack (U2R):** unauthorized access to local super user (root) privileges is an attack type, by which an attacker uses a normal account to login into a victim system and tries to gain root/administrator privileges by exploiting some vulnerability in the victim e.g. buffer overflow attacks. Relevant features: number of file creations and number of shell prompts invoked.

4. **Remote to Local Attack (R2L):** unauthorized access from a remote machine, the attacker intrudes into a remote machine and gains local access of the victim machine. E.g. password guessing Relevant features: Network level features – duration of connection and service requested and host level features number of failed login attempts.

The specific types of attacks are classified into four major categories. The Table 3.7 shows this detail.

Attack Class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf, Teardrop, Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan,Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

TABLE 3.7: Mapping of attack class with attack type

3.2.6 Distribution Details

The Table 3.8 shows the distribution of the normal and attack records available in the various NSL-KDD datasets.

DataSetType	Total No. of					
	Records	Normal Class	DoS Class	Probe Class	U2R Class	R2L Class
KDDTrain+20%	25192	13449	9234	2289	11	209
		53.39%	36.65%	9.09%	0.04%	0.83%
KDDTrain+	125973	67343	45927	11656	52	995
		53.46%	36.46	9.25%	0.04%	0.79%
KDDTest+	22544	9711	7458	2421	200	2754
		43.08%	33.08%	10.74%	0.89%	12.22%
KDDTest-21	11850	2152	4342	2402	200	2754
		18.16%	36.64%	20.27%	1.69%	23.24%

TABLE 3.8: Details of normal and attack data in different types of NSL-KDD data-set

Fig 1 and 2 explains about the analysis of NSL-KDD dataset in detail and shows the number of individual records in four types of attacks for both training and testing.

FIGURE 3.1: Number of instance in training dataset

FIGURE 3.2: Number of instance in testing dataset

3.3 Motivation

Deep learning techniques have enjoyed enormous success in the speech and language processing community over the past few years [68], beating previous state-of-the-art approaches. and played played an important role in the fields of computer vision, natural language processing (**NLP**), semantic understanding, speech recognition, language modeling, translation, picture description, and human action recognition [69, 70], among others.

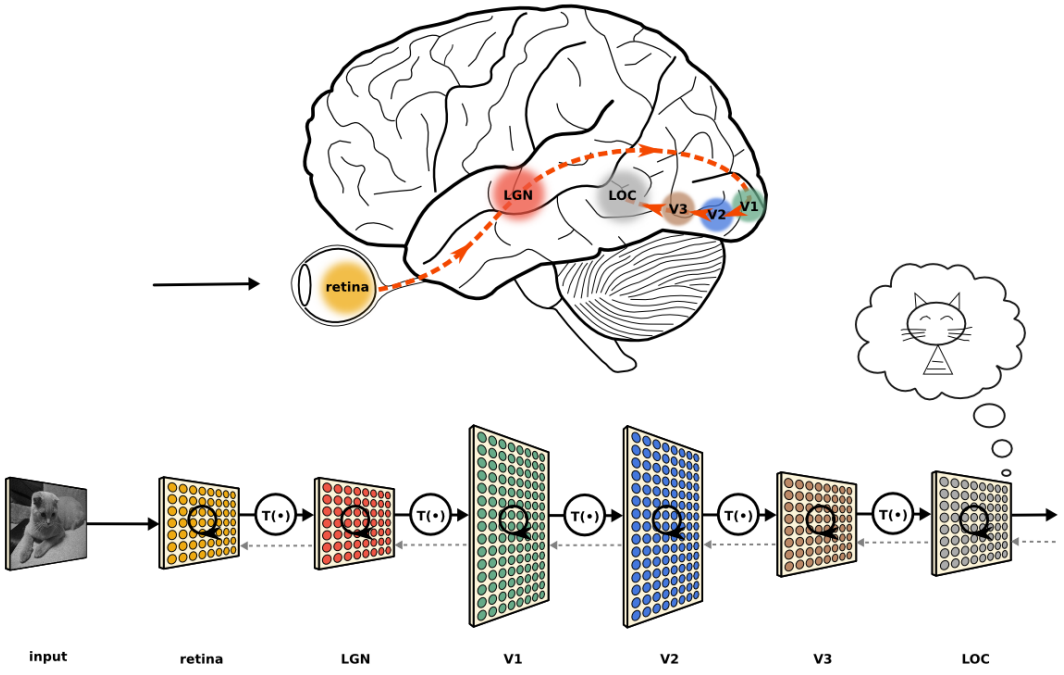

FIGURE 3.3: Example of deep learning

In light of the success of deep learning, we have proposed a deep learning approach for an intrusion detection system using deep neural networks (**DNN**).

We chose a deep neural network as the classifier in our study as it offers a number of advantages over alternative classification approaches, including the need for less formal statistical training, the capability to detect complex non-linear relationships between predictors and the outcomes, the ability to model the interrelationships among the predictor variables and the availability of various training algorithms [71]. The superior performance of deep networks has already been documented in various comparative empirical studies and contests [72], [73], [26].

3.4 Proposed Approaches

We propose two models architectures for IDS *Binary classification model* and *multiclass classification model* based in deep neural network the objective of these models is identifying whether network traffic behavior is normal or anomaly in Binary classification , or a five-category classification problem, identifying whether it is normal or any one of the other four attack types: Denial of Service (DOS), User to Root (U2R), Probe (Probing) and Root to Local (R2L).

We apply preprocessing to the (**NSL-KDD**) dataset, after data preprocessing stage we feed the data to the DNN for building the model and evaluate the performance of the model in binary classification and multiclass classification, beside that we present all the details used to design the network architecture. Forethought Binary classification and multiclass classification are independent of each other.

The Figure 3.3 show to us the global architecture design and the implementation of the intrusion detection system based on deep neural network (**DNN-IDS**).

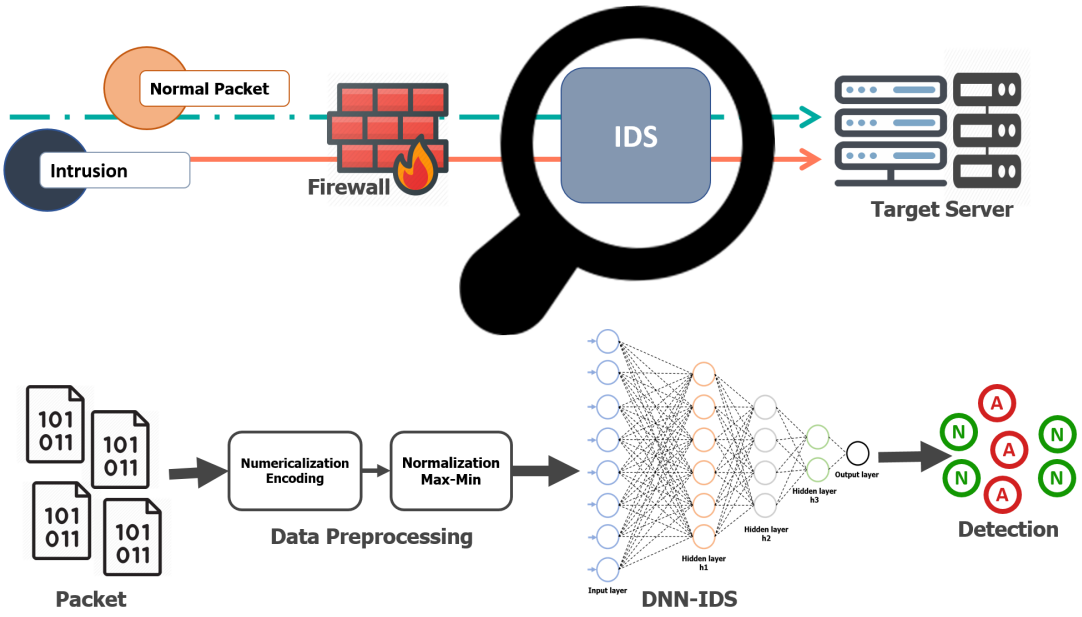

FIGURE 3.4: Architecture of the proposed model

3.4.1 Data Preprocessing

NSL-KDD dataset consists of different data types, the non-numeric values and numeric values we need to converted non-numeric into numeric in *Numericalization* stage. Next bring the dataset into the same range in *Normalization* . These two steps were performed for both NSL-KDD train and test datasets see Figure 3.5.

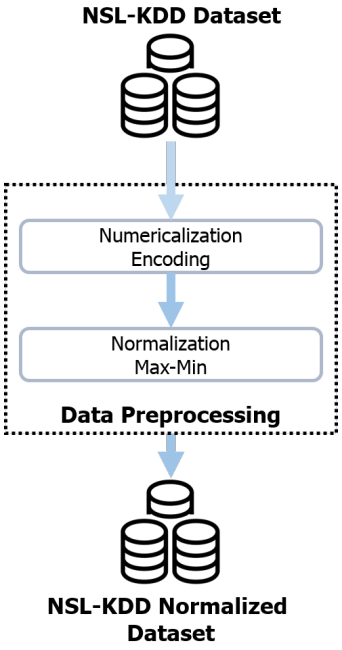

FIGURE 3.5: Data preprocessing model

- **Numericalization:**

  The Deep neural network uses only numerical values for training and testing. Hence a preprocessing stage is needed to convert the non-numerical values to numerical values. Two main tasks in Numericalization are:

  1. *Converting the non-numerical features in the dataset to numerical values,* such as protocol_type, service and flag features, into numeric form. For example, the feature protocol_type has three types of attributes, (tcp, udp, and icmp) and its numeric values are encoded as binary vectors (1,0,0), (0,1,0) and (0,0,1).Similarly, the feature service has 70 types of attributes, and the feature flag has 11 types of attributes. Continuing in this way, 41-dimensional features map into 122-dimensional features after transformation, using OneHotEncoding which is a representation of categorical variables as binary vectors.

2. *Convert the attack types at the end of the dataset into its numeric categories.* 1 is assigned to normal data. 2, 3, 4 and 5 are assigned to DoS, Probe, R2L and U2R attack types respectively.

- **Normalization:**

  Normalization reduces the size of the dataset and more importantly reduces the response time of detection engine by a large extent. Because of the features of the NSL-KDD dataset have either discrete or continuous values, the ranges of the features value were different and this made them incomparable. As a result, the features were normalized by using min-max normalization (Equation 3.1) to map all the different values for each feature to [0, 1] range.

$$x_i = \frac{x_i - Min}{Max - Min} \tag{3.1}$$

In addition, there is one attribute **(num_outbound_cmds)** in the dataset whose value is always 0 for all the records in the training and test data (Table 3-9). We eliminated this attribute from the dataset. The total number of attributes become 121 after performing the above mentioned steps.

DataSetType	**Total No. of**	
	Records	**num_outbound_cmds**
KDDTrain+	125973	0
KDDTest+	22544	0

TABLE 3.9: Detail of num_outbound_cmds attribute in NSL-KDD dataset

3.4.2 Models Architectures

In general a neural network consist of an input layer, an output layer, and one ore more hidden layers, the network is called a deep network. Each layer is composed of multiple neurons, and edges that connect neurons between adjacent layers have weights. The values of neurons (except the neurons in the input layer) and the weights are trained during a training phase.

3.4.2.1 Binary Classification

Binary classification, or Two-class classification, may be the most widely applied kind of machine learning problem. This part is done be using a binary version of the NSL-KDD dataset. This version contains just two different class. So the attack column does only contain the values (Normal, Anomaly). As well we build a model to classify intrusion as normal or anomaly. The following illustrations show to us the step used to build a binary classification model.

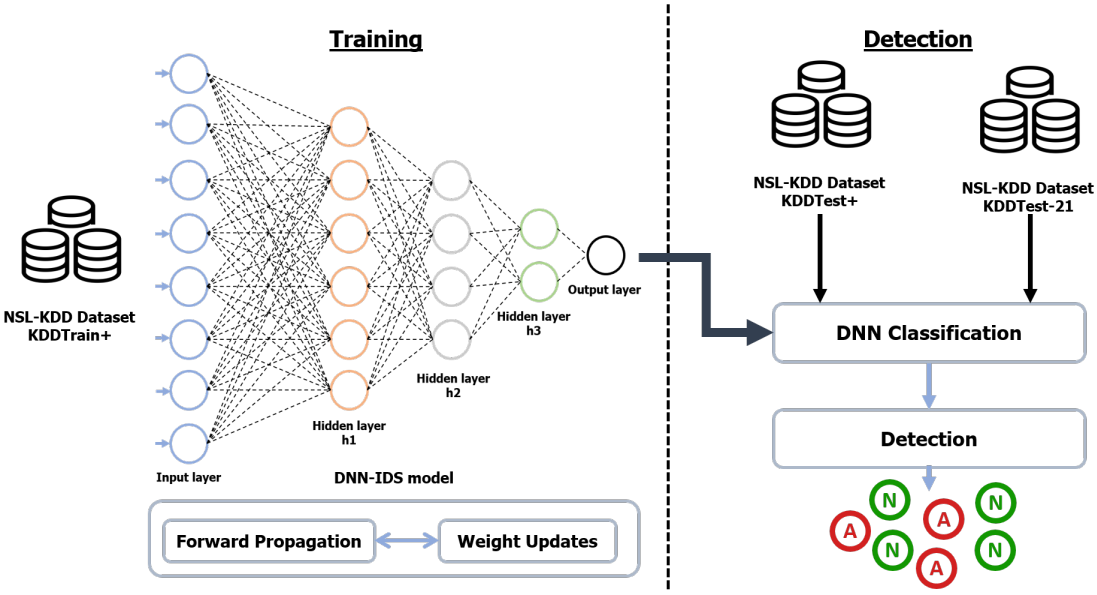

FIGURE 3.6: The overall architecture of DNN-IDS model in binary classification

Network Structure for Binary Classification The neural network was designed using Keras. Keras is a neural networks Application Programming Interface (API) written in Python, it runs on top of either TensorFlow, Theano or Microsoft Cognitive Toolkit (CNTK), which are software libraries for machine learning. The network was designed by stacking network layers on top of each other, as each layer type has its own function in Keras. Our network, including an input layer, three hidden layers and an output layer, is as follows:

layers	Input	H1	H2	H3	Output
Num of neurons	121	256	114	78	1

TABLE 3.10: The number of neurons in each layer in binary classification

Hyperparameters for Binary Classification There are a large number of hyperparameters which have to be defined before training a network. There are certain parameters required for setting up a network and training it. Table 3.11 shows hyperparameter settings which have been used throughout all experiments.

Hyperparameter	Setting
Activation function for neurons	ReLU
Activation function for last layer	Sigmoid
Type of training	Supervised fine-tuning
Weight initialization	Glorot uniform
Dropout rate	20%
Optimizer	RMSprop
Initial learning rate	0.001
Batch size	128
Weight regularizer	None
Lost function	Binary crossentropy error function

TABLE 3.11: Hyperparameter settings in binary classification

Because we are attacking a binary-classification problem, we will end the network with a single unit and a sigmoid activation. This unit will encode the probability that the network is looking at one class or the other.

3.4.2.2 Multiclass Classification

Multiclass classification is when the output of the prediction done by the algorithms can be more then just two classes. In this case, we have build a network to classify intrusion into 5 mutually exclusive classes, the attack column contain five different attacks (DoS, R2L, U2R, Prop, Normal). Because we have many classes, this problem is considered as an instance of multiclass classification, and because each data point should classified into only one category, the problem is more specifically an instance of **single-label, multiclass classification**. The following illustrations show to us the step used to build a single-label multiclass classification model.

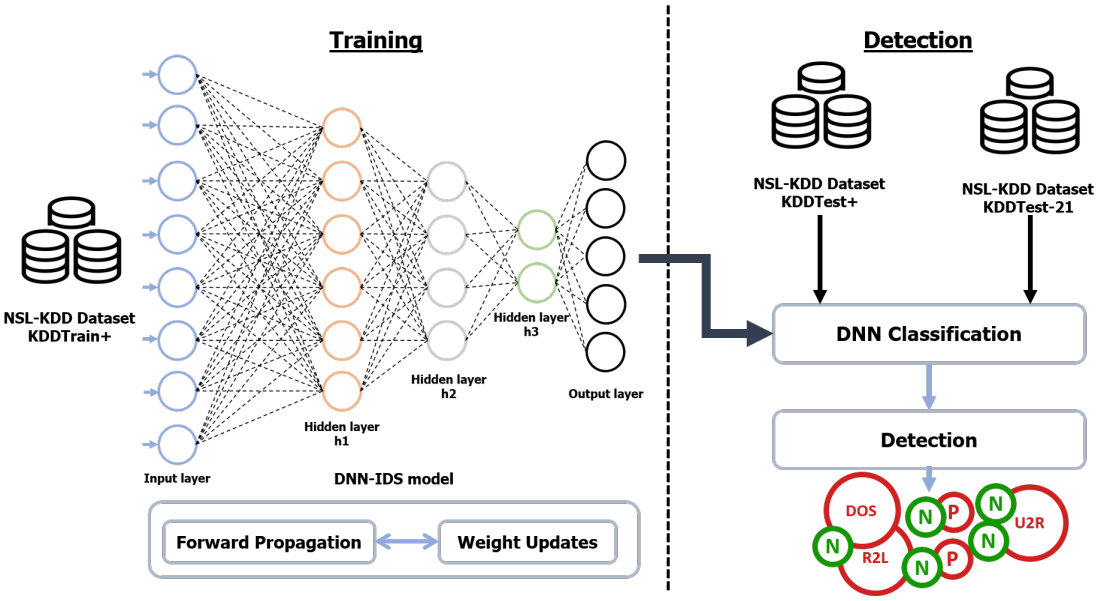

FIGURE 3.7: The overall architecture of DNN-IDS model in multiclass classification

Network Structure for Multiclass Classification This topic-classification problem looks similar to the previous binary classification problem: in both cases, we're trying to classify intrusion. But there is a new constraint here: the number of output classes has gone from 1 to 5. The dimensionality of the output space is much larger. In network like that we have been using, an input layer, three hidden layers and an output layer, is as follows:

layers	Input	H1	H2	H3	Output
Num of neurons	121	2048	1024	512	5

TABLE 3.12: The number of neurons in each layer in multiclass classification

Hyperparameters for Multiclass Classification The following Table 3.12 shows hyperparameter settings which have been used throughout all experiments.

Hyperparameter	Setting
Activation function for neurons	ReLU
Activation function for last layer	Softmax
Type of training	Supervised fine-tuning
Weight initialization	Glorot uniform
Dropout rate	50%, 20%
Optimizer	RMSprop
Initial learning rate	0.0001
Batch size	256
Weight regularizer	None
Loss function	Categorical Crossentropy function

TABLE 3.13: Hyperparameter settings in multiclass classification

Because we are attacking a multiclass classification problem, we will end the network with a 5 units and a softmax activation. It means the network will output a probability distribution over the 5 different output classes, for every input sample, the network will produce a 5-dimensional output vector, where $output_i$ is the probability that the sample belongs to class i. The 5 scores will sum to 1.

3.4.3 Training, Validation and Test Data Sets

Machine learning models need to be trained, selected, and tested on independent data sets to avoid overfitting and assure that the model will generalize to unseen data. Hold-out validation, partitioning the data into a training, validation, and test set, is the standard for deep neural networks. The training set is used to learn models with different hyperparameters, which are then assessed on the validation set. The model with best performance, e.g. prediction accuracy or mean squared error, is selected, and further evaluated on the test set to quantify the performance on unseen data and for comparison to other methods.

Typically we use the data set (*KDDTrain+*) for training the model and the rest (*KDDTest+, KDDTest-21*) for model testing. moreover we use about 60% of (*KDDTrain+*) for training and the remaining 40% for validation in binary classification as well as multiclass classification.

3.4.4 Performance Parameters of IDS

Performance of IDS is measured by various parameters like accuracy, precision, recall, and Fscore. These parameters are used to evaluate the IDS and compare it with other IDS [74]. To calculate the parameters, confusion matrix which is shown in Table-1.1 is used. On the basis of this confusion matrix and formulas shown bellow performance parameters are calculated.

Actual	Predicted	
	Attack	Normal
Attack	True Positive (TP)	False Negative (FN)
Normal	False Positive (FP)	True Negative (TN)

TABLE 3.14: Confusion matrix

As per [75], Accuracy, Precision, Recall and Fscore can be calculated as follows:

- **Accuracy:** the fraction of all correctly predicted among the total amount of predicted shown in (3-2):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

- **Precision:** is the fraction of true positives (TP) among the true and false positives shown in (3-3):

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

- **Recall or True Positive Rate (TPR):** the fraction of true positives (TP) among the true positives and false negatives (TP + FN), as shown in (3.4):

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

- **The F-score:** is a metric to evaluate the accuracy of the predictions (Equation 3.5). For every class, an F-score will be determined. The F-score combines precision (Equation 3.3) and recall (Equation 3.4) featuring a score of 1 as the perfect score:

$$Fscore = \frac{2 * precision * recall}{precision + recall} \tag{3.5}$$

3.5 Conclusion

We have proposed a network intrusion detection system based on deep neural network. Also we use a Deep learning, on NSL-KDD a benchmark dataset for network intrusion. In the next chapter we present the performance of the model in binary classification and multiclass classification and compare it with a previous work. Compared metrics include the accuracy, precision, recall, and f-measure values.

.

Chapter 4

Results and Discussions

4.1 Introduction

In this chapter all of the results from the experiments will be presented. The results will be presented using different kinds of plots, bar charts, tables and some text to explain the different presentations. This chapter should give a overview of how the different experiments performed. The comparison of performance of DNN-IDS with other machine learning methods. The results from the different experiments will be gone through and explained thoroughly.

The objective in this chapter is to present the performance of the model in binary classification and multiclass classification.

4.2 Experimental Environment

In this project, we have used **Keras** one of the most famous library used in production of deep learning and has also been adopted by researchers at large scientific organizations, in particular CERN and NASA [76].

FIGURE 4.1: Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research in deep-learning [76].

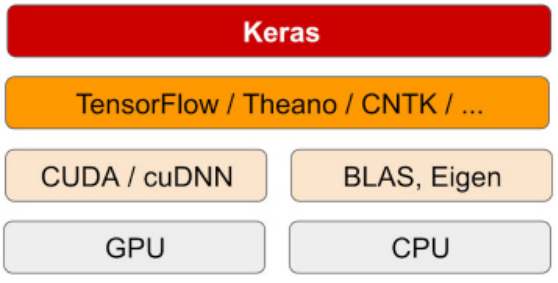

FIGURE 4.2: Software and hardware stack

The experiment is performed on a personal notebook **TOSHIBA SATELLITE PSCG8E** which has a configuration as fallow:

CPU:	Intel(R) Core(TM) i3- 3120M @ 2.50 GHz
Cores:	04
Memory:	06 GiB
OS:	Ubuntu 16.04 LTS 64-bit

TABLE 4.1: Physical machine specifications

As can be seen from Table 4.1, the notebook has CPU power due to it's 4 cores and each and every one of them have 2.50 GHz. Thus the computation or power needed to train and test the deep neural network should not be a problem.

The Operating System(OS) running on the notebook is Ubuntu 16.04 LTS, which Keras framework should work good together with and that should not be a problem.

The notebook also has enough memory available, it has a total of 8 GiB. Which is enough to run and do the processes that needs to be run, in order to do what is intended to do in this project.

4.3 Results and Discussions

4.3.1 Deep Neural Network Binary Classification

4.3.1.1 Evaluation Based on Training data

Figure 4.3 and 4.4 show the training and validation results, the curves indicate there is no overfitting, Also the training curves are closely tracking the validation curves. Additionally The training accuracy increases linearly over time, until it reaches 99.81%, whereas the validation accuracy reach 99.56%. The validation loss reaches its minimum then stalls, whereas the training loss keeps decreasing linearly until it reaches nearly 0.

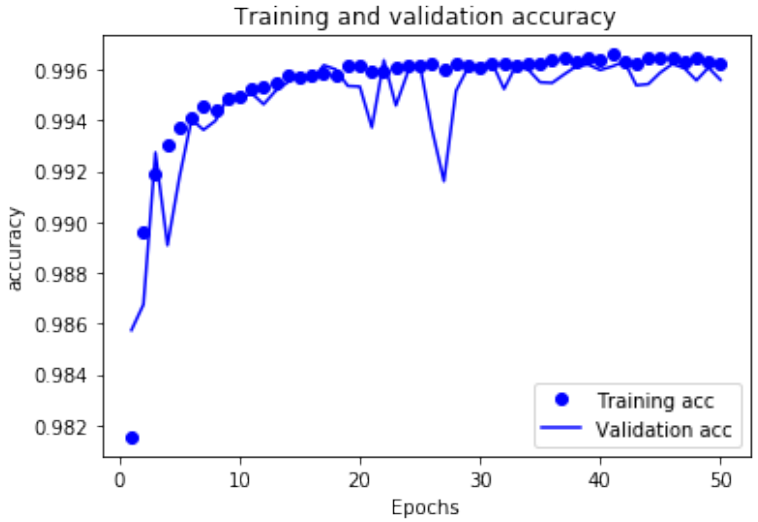

FIGURE 4.3: Training and validation accuracy in binary classification

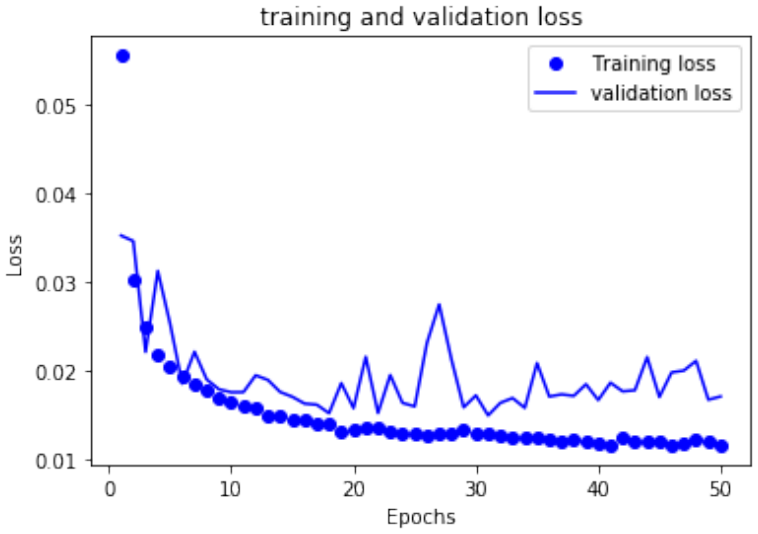

FIGURE 4.4: Training and validation loss in binary classification

4.3.1.2 Evaluation Based on Test Data

Actual	Predicted	
	Attack	Normal
Attack	10062	2771
Normal	756	8995

TABLE 4.2: Confusion matrix binary classification on KDDTest+

Table 4.2 gives a very clear picture on how accurate the DNN model have performed. True positive and True negative values are very high, which indicates that the model

did a good job classifying the test data.

	Accuracy	**Precision**	**Recall**	**F1 score**
KDDTest+	0.8435	0.7636	0.9221	0.8354
KDDTest-21	0.7032	0.3364	0.6524	0.4439

TABLE 4.3: Results of the DNN-IDS in binary classification

Table 4.3 show the accuracy, precision, recall and f1 score with KDDTest+ and KDDTest-21, As observed that DNN perform very well for binary classification and achieved 84.35% accuracy rate in the KDDTest+ and 70.32% in the KDDTest-21.

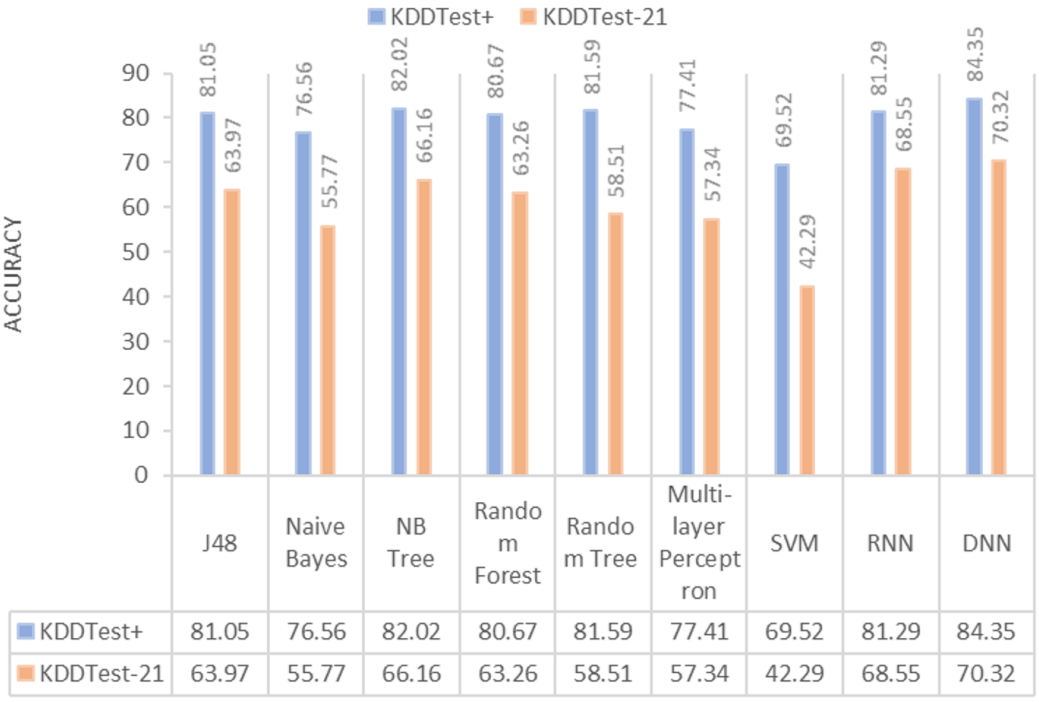

FIGURE 4.5: Performance of DNN-IDS and the other models in the binary classification

The results for the proposed DNN-IDS approach and state-of-the-art approaches are presented in Figure 4.5. We compared the results of our model against state-of-the-art approaches which used the same datasets (NSL-KDD datasets). In [64], the authors have shown the results obtained by J48 (decision tree), Naive Bayesian, NB Tree, Random Forest, Multi-layer Perceptron, Random Tree and Support Vector Machine the best accuracy rate achieved was 82.02% with NB-Tree and the RNN model also give 81.29% in [65], which is the recent literature about deep learning algorithms applied in the

filed of intrusion detection. Obviously, The accuracy achieved using DNN for binary classification outperforms many of the previous work results.

4.3.2 Deep Neural Network Multiclass Classification

4.3.2.1 Evaluation Based on Training data

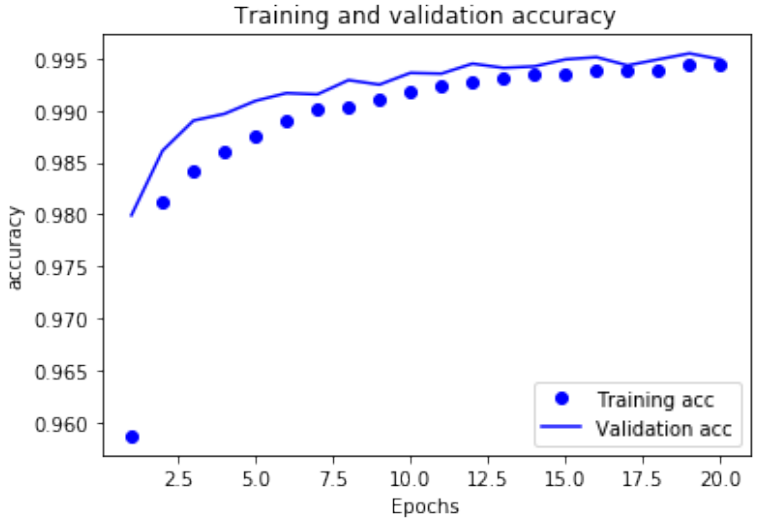

FIGURE 4.6: Training and validation accuracy in multiclass classification

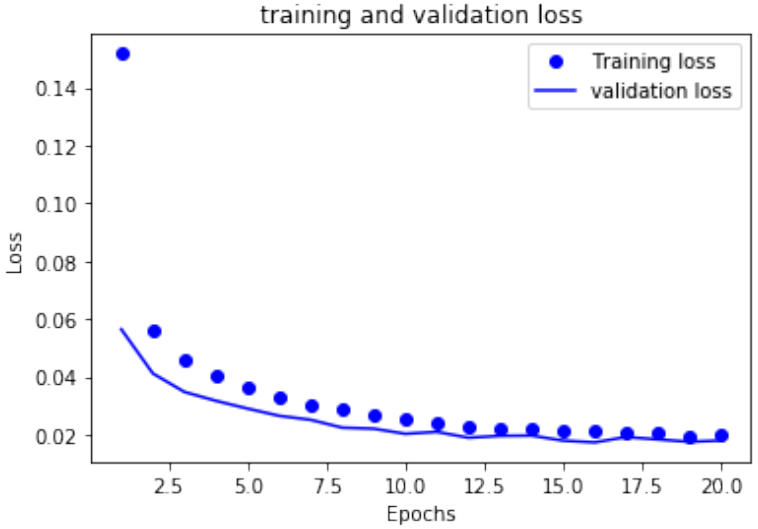

FIGURE 4.7: Training and validation loss in multiclass classification

From figure 4.5 and 4.6 we observe non difference between training and validation which mean the model is not overfitted. Beside The training accuracy increases linearly over time, until it reaches 99.46%, whereas the validation accuracy reach 99.51%, Also The

validation loss reaches its minimum then stalls, whereas the training loss keeps decreasing linearly until it reaches nearly 0.

4.3.2.2 Evaluation Based on Test Data

	Normal	**DoS**	**Probe**	**R2L**	**U2R**
Normal	9427	65	217	0	2
DoS	1114	6314	30	0	0
Probe	510	214	1684	13	0
R2L	2453	1	6	291	3
U2R	175	0	4	4	17

TABLE 4.4: The confusion matrix for DNN-IDS multiclass classification

Table 4.4 displays is how many samples the model classified correctly and incorrectly. The columns are aligned in a way that shows the correctly classified samples in the corresponding columns. So the column Normal which is the first both horizontal and vertical, way to the left in the table, shows the correctly predicted Normal samples. The column that is talked about is the one containing the number 9427. The others in the same row horizontally are the ones that should have been classified as Normal, but were not.

	Precision	**Recall**	**F1-score**	**support**
Normal	0.69	0.97	0.81	9711
DoS	0.96	0.85	0.90	7458
Probe	0.87	0.70	0.77	2421
R2L	0.94	0.11	0.19	2754
U2R	0.77	0.09	0.15	200
Avg/Tottal	0.83	0.79	0.75	22544
Accuracy	**78.65**			

TABLE 4.5: Results of the DNN-IDS in Multiclass classification in KDDTest+

As we observe From table 4.5 and 4.6 they give us the accuracy, recall, precision, and f1 score for each class, Also we notice that DNN perform very well for multiclass classification and achieved 78.65% accuracy rate in the KDDTest+ but did not generalize well in the KDDTest-21 achieved only 59.51% .

	Precision	**Recall**	**F1-score**	**support**
Normal	0.31	0.87	0.45	2152
DoS	0.92	0.74	0.82	4342
Probe	0.87	0.69	0.77	2402
R2L	0.95	0.11	0.19	2754
U2R	0.77	0.09	0.15	200
Avg/Tottal	0.80	0.60	0.59	11850
Accuracy	**59.51**			

TABLE 4.6: Results of the DNN-IDS in multiclass classification in KDDTest-21

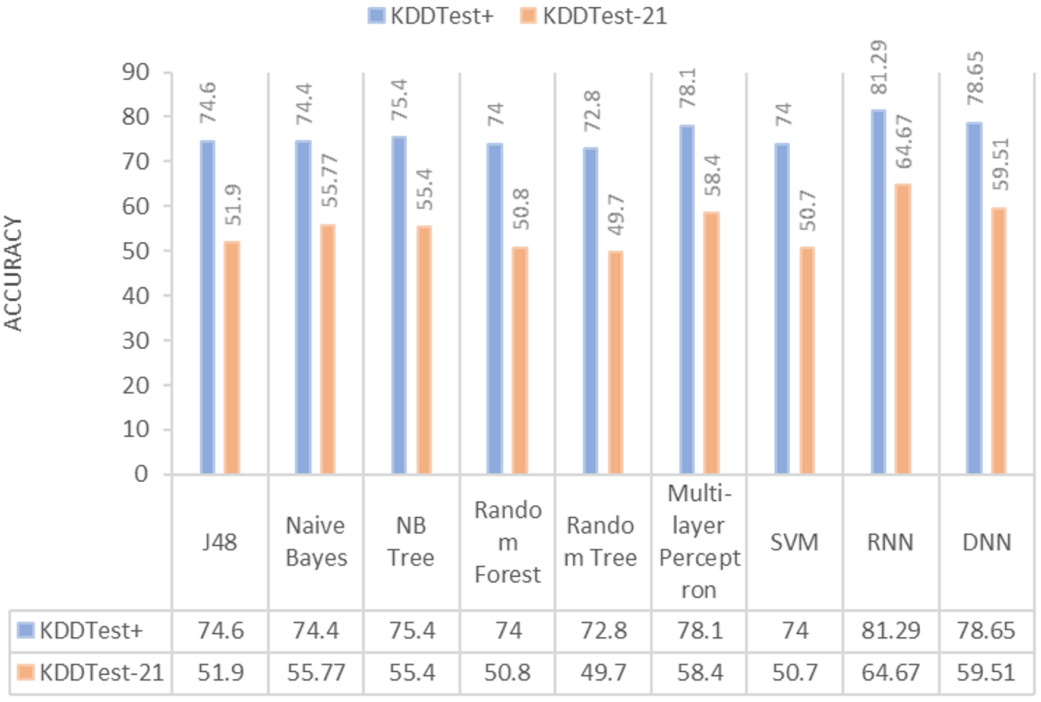

FIGURE 4.8: Performance of DNN-IDS and the other models in the multiclass classification

The experiment shows that the accuracy of the model is 78.65% for the test set KDDTest+ and 59.51% for KDDTest-21, which is better than those obtained using J48, naive bayes, random forest, multi-layer perceptron and the other classification algorithms [64]. But it is not better than RNN which obtained 81.29% in KDDTest+ and 64.51% in KDDTest-21 [65].

4.4 Conclusion

The results presented in this chapter were very encouraging and the experimental results based on the benchmark network intrusion dataset NSL-KDD to evaluate anomaly detection accuracy, show that for both binary and multiclass classification, the intrusion detection model DNN-IDS preform well in KDDTest+ and KDDTest-21 and we get high accuracy rate than other machine learning methods.

Conclusion

The main goal of this work was to investigate the effectiveness of machine learning techniques on detecting malicious traffic and classify it into the corresponding classes of attacks. In our study, we utilized a deep learning technique called deep neural network to build an effective and flexible model for NIDS.

We used the popular benchmark network intrusion dataset NSL-KDD to validate and evaluate the accuracy of our proposal. The NSL-KDD dataset contains various indicators for multiple attacks blended in with normal traffic. There are labels linked to each sample, which makes it possible for the model to differentiate between patterns of attacks compared to normal traffic. We have provided tow implementation of our model using both binary classification and multiclass classification. This was done to observe how well our proposal performed when exposed to binary classification and multiclass classification. We have shown that our DNN-IDS model has high accuracy in both binary and multiclass classification and less false-alarm rates compared with state-of-the-art machine learning methods.

The developed intrusion detection system (DNN-IDS) was intended to be used in a real network environment. However, we did not manage to realize that due to the time pressure.

As concluding remark to the project we may say that it has been truly exciting and fascinating working on this project. It is an open project covering a wide range of subjects and there are a lot of extension possibilities and challenges. One of the main challenging directions is the design of a solution that does not rely on a pre-prepared data such as NSL-KDD and try to work directly on raw traffic packets and detects malicious ones.

Bibliography

[1] Marcus A Maloof. *Machine learning and data mining for computer security: methods and applications*. Springer, 2006.

[2] Belden Menkus. Introduction to computer security. *Computers & Security*, 11(2): 121–127, 1992.

[3] Richard Heady, George F Luger, Arthur Maccabe, and Mark Servilla. *The architecture of a network level intrusion detection system*. University of New Mexico. Department of Computer Science. College of Engineering, 1990.

[4] Denning ED. An intrusion-detection model. *IEEE Transactions on Software Engineering 1987*, 1987.

[5] Yingbing Yu. A survey of anomaly intrusion detection techniques. *Journal of Computing Sciences in Colleges*, 28(1):9–17, 2012.

[6] Stephen E Smaha. Haystack: An intrusion detection system. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44. IEEE, 1988.

[7] Phillip A. Porras and Alfonso Valdes. Live traffic analysis of TCP. IP Gateways. *To appear in Internet Society's Networks and Distributed Systems Security Symposium.*, 1998.

[8] Adebayo D. Akinde Saidat Adebukola Onashoga. A Strategic Review of Existing Mobile Agent- Based Intrusion Detection Systems. *Issues in Informing Science and Information Technology Volume 6*, 2009.

[9] E Lundin and E Jonsson. Survey of research in the intrusion detection area. Technical report, Technical report 02-04, Department of Computer Engineering, Chalmers University of Technology, Göteborg January 2002, http://www. ce. chalmers. se/staff/emilie/papers/Lundin_survey02. pdf.

[10] Paul Innella. An introduction to ids. URL https://www.symantec.com/connect/articles/introduction-ids[AccessedinFebruary2018].

[11] Mark Crosbie and Gene Spafford. Active defense of a computer system usingautonomous agents. *Technical Report 95-008, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398*, 1995.

[12] G. Kolaczek A. Grzech K. Juszczyszyn, N. T. Nguyen and A. Katarzyniak. Agent-based approach for distributed intrusion detection system design. *Lecture Notes in Computer Science, pp. 224–231*, 2006.

[13] Y. P. Jiang Z. Y. Cai, Y. Gan and W. J. Zhong. Development of intelligent intrusion detection based on biosimulation,. *Journal of Zhengzhou University of Light Industry,*, 2010.

[14] Yuebin Bai and Hidetsune Kobayashi. Intrusion detection systems: technology and development. In *Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on*, pages 710–715. IEEE, 2003.

[15] Yichi Zhang, Lingfeng Wang, Weiqing Sun, Robert C Green II, and Mansoor Alam. Distributed intrusion detection system in a multi-layer network architecture of smart grids. *IEEE Transactions on Smart Grid*, 2(4):796–808, 2011.

[16] Eric White and Patrick Turley. System and method for providing a secure connection between networked computers, April 23 2013. US Patent 8,429,725.

[17] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics*, 44(1):66–82, 2014.

[18] Francois Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[20] Chris Bishop, Christopher M Bishop, et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[21] M Bishop Christopher. *PATTERN RECOGNITION AND MACHINE LEARNING.* Springer-Verlag New York, 2016.

[22] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[24] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[25] Nils J Nilsson. *The quest for artificial intelligence.* Cambridge University Press, 2009.

[26] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[27] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[28] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.

[29] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[30] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[32] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[33] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

[34] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[35] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[36] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[40] Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013.

[41] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Dropout distillation. In *International Conference on Machine Learning*, pages 99–107, 2016.

[42] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[43] Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. Rnndrop: A novel dropout for rnns in asr. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 65–70. IEEE, 2015.

[44] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[45] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation*, 8(3):643–674, 1996.

[46] Kam Jim, Bill G Horne, and C Lee Giles. Effects of noise on convergence and generalization in recurrent networks. In *Advances in neural information processing systems*, pages 649–656, 1995.

[47] Kiyotoshi Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

[48] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[49] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[50] Christopher M Bishop. Regularization and complexity control in feed-forward networks. 1995.

[51] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[52] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[53] Matthew D Zeiler, M Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.

[54] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[56] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[57] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[58] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[59] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4580–4584. IEEE, 2015.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[61] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. corr abs/1505.00387, 2015.

[62] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. Highway long short-term memory rnns for distant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5755–5759. IEEE, 2016.

[63] Mark Robins. The future of deep learning: Challenges & solutions. In *Proceedings of the Computing Frontiers Conference*, pages ii–ii. ACM, 2017.

[64] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[65] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5: 21954–21961, 2017.

[66] Sapna S Kaushik and PR Deshmukh. Detection of attacks in an intrusion detection system. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 2(3):982–986, 2011.

[67] One of Canada's top universities Leading discovery and innovation since 1785. Unb. http://www.unb.ca/cic/datasets/index.html, 2015.

[68] D. Yu G. E. Dahl A. Mohamed N. Jaitly A. Senior V. Vanhoucke P. Nguyen T. N. Sainath G. Hinton, L. Deng and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82–97*, 2012.

[69] X. Wang X. Peng, L. Wang and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice,. *Computer Vision and Image Understanding, vol. 150, pp. 109-125*, 2016.

[70] W. Li W. Nie, A. Liu and Y. Su. Cross-view action recognition by cross-domain learning,. *Image and Vision Computing, vol. 55, no. P2, pp. 109-118,*, 2016.

[71] Jack V Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231, 1996.

[72] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[73] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[74] Bhushan Trivedi Bhavin Shah. Reducing Features of KDD CUP 1999 Dataset For Anomaly Detection Using Back Propagation Neural Network. *IEEE International Conference on Advanced Computing And Communication Technologies*, 2015.

[75] Koral Ilgun. Ustat: A real-time intrusion detection system for unix. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, pages 16–28. IEEE, 1993.

[76] François Chollet et al. Keras. https://keras.io, 2015.