Intrusion Detection with Artificial Neural Networks

Moazzam Hossain

February 10, 2004

Abstract

The purpose of this work is to design, implement and evaluate an anomaly based network intrusion detection system. Intrusion detection system is a detection mechanism that detects unauthorized, malicious presence in the computer systems. An anomaly based system is divided into two phases: learning and detection. In learning phase, the system learns about the normal users' or system's behavior. In the detection phase, system detects the intruders by matching their behavior with that of normal users'. It is obvious that the intruders' behavior is different than that of normal users'.

In this system, we use neural network for learning about normal users' behavior. Neural network learns by training. We use a special type of neural network called backpropagation neural network for this purpose. The neural network learns about the normal users' behavior form the network traffic that only contains information about normal users. When the learning is over, the system is tested with the network traffic that contains both attacks and normal data.

We have tested the system performance by using a simulated computer network. We divide the training process of neural network into three different approaches. The neural network is trained with huge, not so huge and small amount of training data. We have tested the detection capability of the system with huge as well as small amount of test data. It is seen from the performance analysis that the system performs well when trained with small amount of data.

We have compared our system performance with that of two other research works. In our work, the overall detection rate for normal user is 100% and for attacks is 98% which is considerably better than that of other two works where they have attack detection rate of 24% and 86% respectively.

Keywords: Computer Security, Intrusion Detection, Artificial Neural Networks, Anomaly-based system, Backpropagation.

Acknowledgements

I would first like to show my heartiest gratitude to my supervisor Dr Christian Damsgaard Jensen, Associate Professor, IMM, as it is obvious that without his endless support, encouragement and assistance it would not be possible to accomplish this dissertation.

I am also grateful to Associate Professor Niels-Ole Christensen for his assistance about neural network.

I thank everybody at IMM for their assistance and cooperation. Additional thanks to Soumya Dutta for checking the report.

And lastly, thanks to Danish government for letting me study at DTU without any tuition fee.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Systems Engineering at Technical University of Denmark(DTU), Lyngby, Denmark.

The work has been carried out by Moazzam Hossain(s011039) from June 2003 to December 2003.

Thesis supervisor is Associate Professor Christian Damsgaard Jensen, Computer Science and Engineering, Institute of Mathematical Modeling(IMM), Technical University of Denmark(DTU).

Contents

1 Introduction **1**

1.1 Background 1

1.1.1 Computer Security and Intrusion Detection 1

1.1.2 Artificial Neural Networks 2

1.1.3 Related works 2

1.2 Problem Definition 3

1.3 Methodology 3

1.4 Results 3

1.5 Outline of the report 4

2 State of The Art **5**

2.1 Introduction 5

2.2 Intrusion Detection Systems(IDS) 5

2.2.1 IDS in General 5

2.2.2 Classification of IDS 8

2.2.3 Use of Artificial Intelligence in IDS 12

2.3 Artificial Neural Networks 13

2.3.1 Neural Networks in General 13

2.3.2 Learning Methods 15

2.3.3 Connection Structures 16

2.3.4 Types of Neural Networks 18

2.4 Other Works on IDS and Artificial Neural Networks 21

2.4.1 Neural Network Intrusion Detector(NNID) 21

2.4.2 Intrusion Detection Using Neural Networks and Support Vector Machines 22

2.4.3 Network-Based Intrusion Detection Using Neural Networks 22

2.4.4 A Neural Network Based Intelligent Intrusion Detection System . . . 23
2.5 Summary . . . 24

3 Design **25**
3.1 Introduction . . . 25
3.1.1 Design Goals . . . 25
3.1.2 Overview . . . 26
3.2 Collecting and Preprocessing Input data . . . 27
3.3 Training . . . 29
3.3.1 Network Architecture . . . 30
3.3.2 Retrieving input data from file . . . 30
3.3.3 Training the network . . . 31
3.4 Detection . . . 31

4 Implementation **32**
4.1 Introduction . . . 32
4.2 Input Data Preprocessing . . . 32
4.3 Training . . . 35
4.3.1 Retrieving input data from file . . . 36
4.3.2 Backpropagation Algorithm . . . 36
4.3.3 Training the network . . . 38
4.4 Detection . . . 40
4.4.1 Forward Propagation Algorithm . . . 40
4.4.2 Detection and Response . . . 41
4.5 Management of Files in Our System . . . 42

5 Evaluation **43**
5.1 Introduction . . . 43
5.2 Training and Test data . . . 43
5.2.1 DARPA Data Sets . . . 43
5.2.2 Training Data Sets . . . 44
5.2.3 Test Data Sets . . . 44
5.3 Training and Experimental Results . . . 46
5.3.1 Termination Conditions . . . 46

5.3.2 First Approach 47
5.3.3 Second Approach 48
5.3.4 Third Approach 48
5.3.5 Discussion on Results 50
5.3.6 Comparison with Other Related Works 51
5.4 Summary 51

6 Conclusion **53**
6.1 Introduction 53
6.2 Difficulties Encountered and Limitations 53
6.3 Achievements 53
6.4 Future extension 54

A Instructions for the software **56**
A.1 Contents of the CD 56
A.2 How to run the program 56

List of Figures

2.1 Components of Intrusion Detection Framework. 7
2.2 A taxonomy of IDS proposed by Debar et al. 9
2.3 Revised taxonomy of IDS proposed by Debar et al. 10
2.4 A Biological Neuron 14
2.5 An Artificial Neuron 15
2.6 A schematic diagram of Supervised Learning. 16
2.7 A schematic diagram of Unsupervised Learning. 17
2.8 Perceptron: An example of Feedforward structure. 17
2.9 Hopfield net: An example of Feedback structure. 18
2.10 A taxonomy of Neural Networks 19
2.11 Boolean functions AND and OR are linearly separable, XOR is not 19

3.1 Block diagram of system overview 27
3.2 Sample sessions from network traffic 28
3.3 Converting a session into binary form 29
3.4 Neural Network Architecture 30

4.1 Input vector, Bias, output and the neural net 36

5.1 Comparison of performance of related works 51

List of Tables

2.1 Different types of Neural Networks and their characteristics 24

5.1 Parameters of neural network 46
5.2 Experiment results for the first approach 47
5.3 Detection rate of the first approach 47
5.4 Experiment results for the second approach 48
5.5 Detection rate of second approach 48
5.6 Experiment results for the third approach 49
5.7 Detection rate of third approach 49
5.8 Experiment results for third approach with huge data 50
5.9 Approximate time taken for training the neural net 50

Chapter 1

Introduction

The detection methods of intruders in the computer networks have drawn attention of many researchers in recent years. An intrusion detection system is the attempt of detecting intruders in a computer system or network. Keeping pace with the increasing amount of use of sophisticated computing tools in almost every aspects of life, risks and malicious intrusions are also increasing. It is very important to design security mechanism to prevent all these malicious activities to the system resources and data. However, it is not possible, at present, to have a complete prevention. And hence, designing of intrusion detection systems becomes necessary for detecting the unprevented attacks so that the appropriate actions can be taken to repair the damages.

Although it has not been possible yet to design a perfect intrusion detection system, it still brings some hope of decreasing the damages.

1.1 Background

In this section, a brief discussion will be made on Computer Security, Intrusion Detections and Artificial neural networks. Elaborate discussion will be made in the next chapter. A brief mention will also be made to the existing research works done for intrusion detection.

1.1.1 Computer Security and Intrusion Detection

Computer security is defined as *technological and managerial procedures applied to computer systems to ensure the availability, integrity and confidentiality of information managed by the computer system* [1]. It can be divided into three areas namely prevention, detection and reaction [2]. Intrusion detection falls into the second category. As Edward [3] defines, *Intrusion Detection is the process of identifying and responding to malicious activity targeted at computing and networking resources.* Numerous techniques and controls are normally adopted to prevent the computing system from unauthorized and malicious attacks. If somehow all these fail, Intrusion detection acts as a next line of security for the system by attempting to detect those attacks. There are two types of intrusion detection systems: signature based and heuristic or anomaly based. In signature based system, intruders are detected from the previous known attacks but in anomaly based systems, intruders are detected from the unusual behavior of the user. From another viewpoint,

intrusion detection systems can again be divided into two categories: Network based and Host based, depending on whether the intruders are being detected in a network or in an individual computer system.

1.1.2 Artificial Neural Networks

Artificial neural network is an information processing paradigm that is inspired by the biological nervous systems, such as the brain, process information [5]. It tries to represent the physical brain and thinking process through electronic circuit or software. Artificial neural network is the network of individual neurons. Each neuron in a neural network acts as an independent processing element [6]. Like human or other brain, neural networks also learn by example or training, they cannot be programmed to perform a specific task [5]. And thus, it can be configured for any specific application through a learning process. Neural networks perform very successfully for recognizing and matching complicated , vague, or incomplete patterns. The most successful application of neural network is classification or categorization and pattern recognition [7].

There are two different learning methods for the neural networks: supervised and unsupervised. In supervised learning method, a neural network learn from the presence of both input and desired output data. On the other hand, in unsupervised learning method, a network learns only from the presence of input data.

1.1.3 Related works

So far, research has been carried out by numerous researchers for designing both anomaly based and signature based intrusion detection systems. For each of these intrusion detection types, there are two basic techniques used to detect intruders: Pattern matching and Protocol analysis [11]. However, a number of techniques has been adopted by the researcher to detect the intruders for both anomaly and signature based systems.

For anomaly detection, techniques like Machine Learning and Data Mining, Computer Immunological approach, Specification-Based Methods etc. have been used. Machine learning and Data mining techniques includes Time-Based Inductive Machine, Instance Based Learning, Neural Networks, Audit Data Analysis and Mining etc. Among those techniques, Neural Networks are used for modeling system as well as user behavior and Data mining techniques are applied to discover abnormal patterns in large amount of audit data(e.g. network traffic collected by tcpdump). Computer Immunological approach is based on an analogy of the immune system's capability of distinguishing *self* from *non-self*. In the Specification-Based Methods, the idea is to use ordered sequences of execution events to specify the intended behaviors of concurrent programs in distributed system [10, 13].

On the other hand, for misuse detection, techniques like Rule-based Languages, State Transition Analysis Technique, Colored Petri Automata, Automatically Build Misuse Detection Models, Abstraction Based Intrusion Detection etc. have been used. In Rule-based Languages, the patterns of known attacks are specified as rule sets, and a forward-chaining expert system is usually used to look for signs of intrusions. State Transition Analysis Technique can be adopted to facilitate the specification of the patterns of known attacks [10, 13].

1.2 Problem Definition

This project describes an artificial neural network based system for network anomaly detection. The system will take network traffic data to analyze and classify the behavior of the authorized users and recognize the likely attacks therefrom. In this project, only offline detection of attacks is to be done. The task will be performed by

1. Inputing a number of input data which represent network sessions for several weeks, to the neural network.
2. Classifying the users' behavior from the input data using back propagation neural network.
3. Converting the output of the neural network as normal or attack.
4. Notifying the likely attacks.

1.3 Methodology

This thesis work is carried out in the following steps:

1. Studying literature on intrusion detection, computer networks emphasizing on network traffic data, and artificial neural networks.
2. Survey of existing systems for intrusion detection.
3. Analysis and design of the proposed system for intrusion detection.
4. Implementation of the proposed design by developing a prototype of the system.
5. Carrying out experiment on the prototype to analyze and evaluate the system.

1.4 Results

The system performance has been tested and satisfactory results have been obtained. Different approaches are made to check the system performance. The system neural network is trained with huge training data and also with small amount of training data. In three different approaches, neural network is trained with 732656, 234190 and 450 different input patterns respectively. One input pattern is the set of input data to the neural network. In the first two training approaches, correct attack detection rate is 56% and 82.5% respectively. The false alarm rate(i.e. when a normal user is mistakenly detected as an intruder by the system) for both cases is 0%. In the last approach where system is trained with only 450 training pattern, the evaluation succeeds with a correct attack prediction rate of 98% for a small amount of test data and 95% for huge test data. The normal user behavior detection rate is 100% in all cases. It means, the system performs without raising any false alarm, the main drawback of anomaly based intrusion detection systems.

1.5 Outline of the report

In the next chapter, a study of state of the art of intrusion detection systems and artificial neural network will be presented. Some existing systems on intrusion detection will also be surveyed.

Our proposed approaches to solve the problem will be presented in chapter 3. In this chapter, the objectives and justifications of the approaches will also be emphasized. The steps followed by our system will be discussed and theoretical basis will be provided where necessary.

Chapter 4 details on implementation. Algorithms along with necessary data structures and issues regarding the neural network used in our system will be discussed here.

Detail experiments of our system will be presented in chapter 5. Here, data and their characteristics will also be discussed. An evaluation of the result will conclude the chapter.

In the last chapter, achievements of the project as well as difficulties encountered will be mentioned. Some possible future extension of the system will also be proposed in this chapter.

Chapter 2

State of The Art

2.1 Introduction

Day by day, dependence on computers are increasing everywhere in the world. More and more computers are connected to the Internet every day and keeping pace with the development of computer technology, security is becoming a major concern. Much effort has been given to prevent the intruders from the computer systems as well as from the networks. Some preventive methods are working quite well but intruders remain a big threat to the computer security. And thus, research trends to pay attention not only to prevention but also to detection. Intrusion Detection System, hence, becoming a favorite research topic. In order to understand the process of intrusion detection, knowledge of the fields related to this ongoing research should be gathered. Therefore, to get deep insight into the research, we studied the following fields of study: Intrusion Detection Systems and Artificial Neural Networks. The detailed description of these topics as well as the *backpropagation neural networks*, which we used to develop our intrusion detection system, is given in this chapter. Moreover, some pictorial representation is also given for the quick realization of the respective topics.

2.2 Intrusion Detection Systems(IDS)

An Intrusion Detection System is a device (e.g. a computer) or a piece of software, that monitors activity to identify malicious or suspicious events. A general presentation of intrusion detection systems including why they are needed and how they work is given in the following subsection. A brief mention about what types of functions are performed by numerous intrusion detection systems and what are their problems is also added. A classification along with a taxonomy is given in a separate subsection. Use of artificial intelligence for IDS is described at the end of this section.

2.2.1 IDS in General

In general, IDS works like a burglar alarm system. It attempts to detect the intruders when all preventive devices fail in a computer system. Like in real life, we may try to protect our possessions, for instance our house, with a powerful and strong lock on the

door when we are on vacation, but through the broken window or some other way the intruders may get in even without using the door at all. Or, they may be able to make a duplicate key to get in through the door. We know, all these things may happen and that is why we use the alarm system, which becomes active by the unauthorized presence in the house and security men come and take necessary action. Similarly, in computer system, intruders may penetrate through the firewall and other preventive materials. In this type of situation, IDS works as the above mentioned alarm system by detecting the presence of unauthorized users'.

Why IDS

The underlying reasons why we use IDS are mainly because we want to protect our data and systems integrity from the intruders[14]. These intruders can be unauthorized outsiders or authorized insiders trying to do unauthorized job in the system. Real world security includes prevention, detection and response. If the prevention mechanisms were perfect, we wouldn't need detection and response. But no prevention mechanism is perfect, which is especially true for computer networks.

The reasons behind this imperfection are very simple. Normally, the protection mechanisms used for security aspects are administrative passwords, cryptographic methods, user access control etc. But the password can be lost and later be found by an intruder, or it can be cracked. Cryptographic methods can be broken. Moreover, it has been seen that the relationship between the level of access control and user efficiency is an inverse one, which means that the stricter the mechanisms, the lower the efficiency becomes[15]. And even a truly secure system is vulnerable to be abused by the insiders who abuse their privileges. As a matter of fact, nowadays, it is not possible to protect the data and systems integrity from the intruders using the usual authentication systems like ordinary password and file security mechanism or even access control mechanism like firewall.

Adequate system security is of course the first step of ensuring data protection. For instance, if there is no administrative password and so on, it makes no sense to connect a system to the Internet and expect that nobody will break into it. Similarly, it is important to prevent the access to the vulnerable files and authentication databases(e.g. /etc/password or /etc/shadow files) in the system except by the authorized system administrators. Furthermore, firewall can always be put in place to enforce access control. But as stated above, all these preventive mechanisms are not perfect. And hence, comes the necessity of intrusion detection.

Intrusion detection takes all these one step further. It can provide an extra layer of protection to the system. When all the preventive systems like above mentioned fail to prevent the intruders, intrusion detection system tries to detect them and inform the administrator for taking appropriate action.

Intrusion Detection Frameworks

An intrusion detection system consists of different discrete components. These components communicate among them via message passing. According to the Common Intrusion Detection Framework(CIDF) [16], four basic components in an intrusion detection are: Event generators, Event analyzers, Event databases and Response units(See figure 2.1).

All these components exchange data among themselves in a generalized form called gido(generalized intrusion detection objects) which is nothing more than an encoding of the fact that some particular occurrence happened at some particular time, or some conclusion about a set of events, or an instruction to carry out an action.

The event generator obtains the raw events from the larger computational environment(e.g. tcpdump or C2 audit trails) outside the intrusion detection system. After obtaining the raw data, it provides them(either unchanged or reduced form) to the other relevant components of the system in a gido format.

The event analyzer analyzes the events obtained from the event generator and return new gidos either for taking action or for preserving or for both. Normally it checks whether the event resembles the signature of the previous known attack or it is an anomaly.

The event database stores gidos for future use. It may store the events generated by the event generator or the analyzed event from the event analyzer.

The Response units take gidos that require some sorts of action(e.g. raising an alarm) from other components and react accordingly.

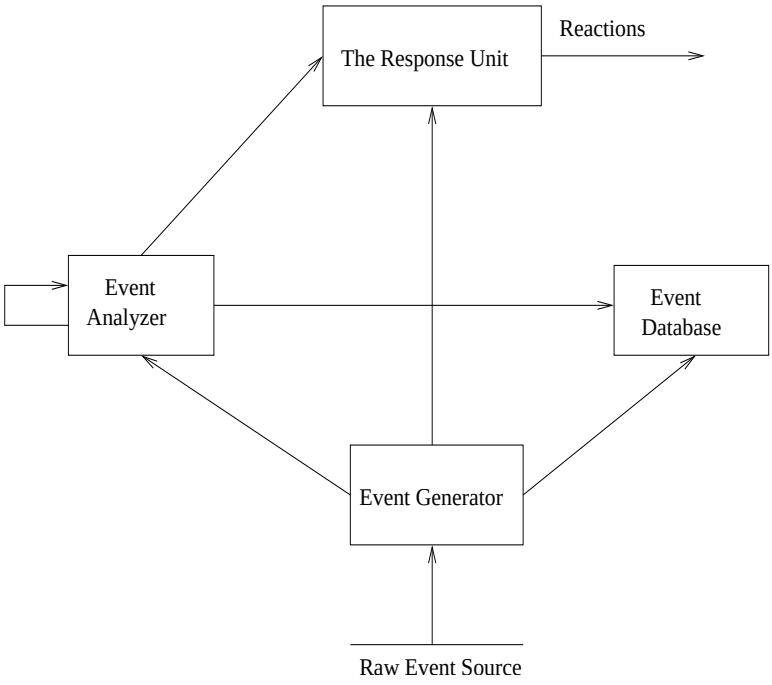

Figure 2.1: Components of Intrusion Detection Framework.

Functions of Intrusion Detection Systems

IDSs perform numerous functions. Following is a list of functions performed by IDSs, although, no single IDS performs all of these functions[17, 20]:

- Intrusion Detection Systems continuously monitor the network traffic and behavior of computers and detect irregular and suspicious activities.

- They can audit the system configurations for vulnerabilities and misconfigurations.
- They can recognize known attack patterns in system activity with the help of a continuously updated database of known attacks.
- They can identify abnormal activities through statistical analysis.
- They immediately detect malicious activity and perform defensive tasks.
- They can detect activities originating from the external network which are allowed by the firewall configuration(such as web browsing) but in fact are harmful (such as exploiting a bug of the web browser).
- They can also identify unauthorized access attempts by internal users.

False Positive and False Negative

Intrusion Detection systems are not perfect systems, so mistakes may happen in them. Two considerable mistakes of intrusion detection systems are *False Positive* and *False Negative*. Normally, when the IDS detects an ongoing attack in the computer system, it raises alarm for taking action by the administrator. False Positive is a state where IDS raises an alarm of attacks for something that is not really an attack. If too many false positives occur, it makes the administrator less confident about the alarms and hence, there is a possibility of ignoring the real attack by the administrator. On the other hand, False Negative is a state where IDS does not raise an alarm for a real attack. This means, a real attack is passing by without the action being taken. This must be very dangerous for the system as the real attacks are completely being overlooked.

2.2.2 Classification of IDS

A few number of taxonomies of IDS proposed by different researchers at different time have been described in this section. Based on the taxonomy, a few number of IDSs, in the context of this work, have also been described.

Taxonomy of IDS

The development of taxonomies of IDSs is not very old and probably the taxonomy proposed by Debar et al.[22] is one of the first real IDS taxonomies. In this taxonomy they have classified the intrusion detection system in terms of detection method, behavior on detection, audit source location and usage frequency(see figure 2.2 for details). The detection method is used to differentiate IDSs into *Behavior based* and *Knowledge based*. On the other hand, using audit source location, IDSs can be distinguished from host log files e.g. *Host based IDS* and from network packets e.g. *Network based IDS*.

The taxonomy proposed by Debar et al. is again extended and refined by Debar et al.[23] themselves and by Axelsson[24]. In the revised version, Debar et al. takes detection paradigm into account in addition to the previous elements of classification like detection method, behavior on detection etc. (see figure 2.3 for the revised taxonomy of Debar et al.). If the detection paradigm of an IDS is state based, the IDS tries to recognize a given

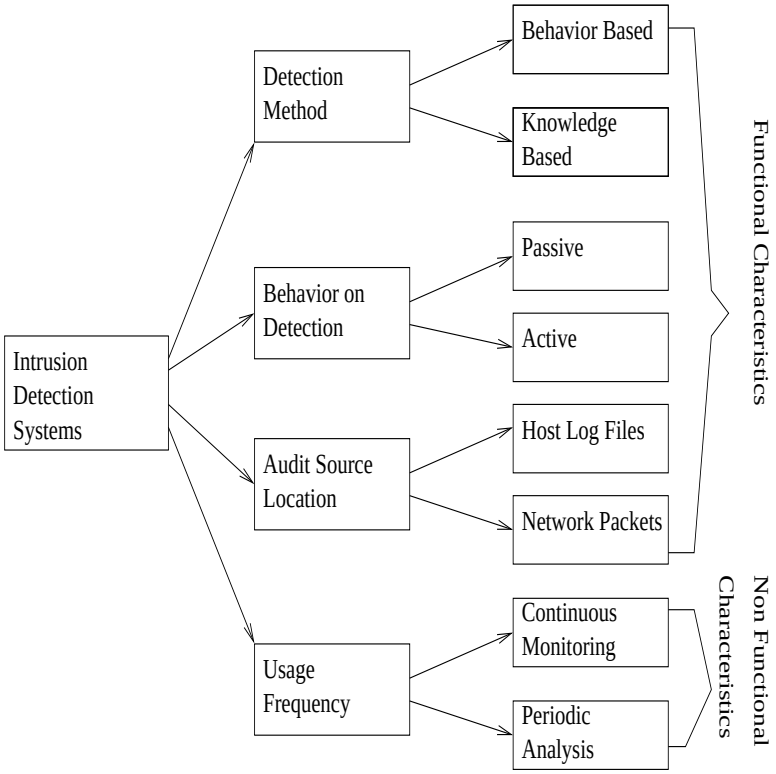

Figure 2.2: A taxonomy of IDS proposed by Debar et al.

system state as being an error state or as being a failure state. Transition based IDSs monitor a system for any state transition that represents an attack or an intrusion [25]. Moreover, the audit source location is modified by adding new categories like application log files and IDS sensor alerts.

Signature based IDS

This type of intrusion detection system contains a database of known vulnerabilities. It monitors traffic and seeks a pattern or a signature match. This means, it operates in much the same way as a virus scanner, by searching for a known identity or signature for each specific intrusion event. It can be placed on a network to watch the network vulnerabilities or can be placed on a host.

A signature-based IDS examines ongoing traffic, activity, transactions, or behavior for matches with known patterns of events specific to known attacks and it raises alarms only when the so called match is found. This is why the number of false positive alarms is comparatively less in signature based IDS. If the system is not entirely new i.e. when it has an up to date database of signatures of known attacks, this technique works extremely well.

On the other hand, it is always difficult to gather knowledge about known attacks and keeping the database up-to-date with new vulnerabilities as lots of new attacks are generating almost every day. It means, it usually fails to detect the new attacks. Moreover, it is a common technique to create new attacks by making some changes in the known attacks rather than making a completely new one[34]. If signature definitions are too spe-

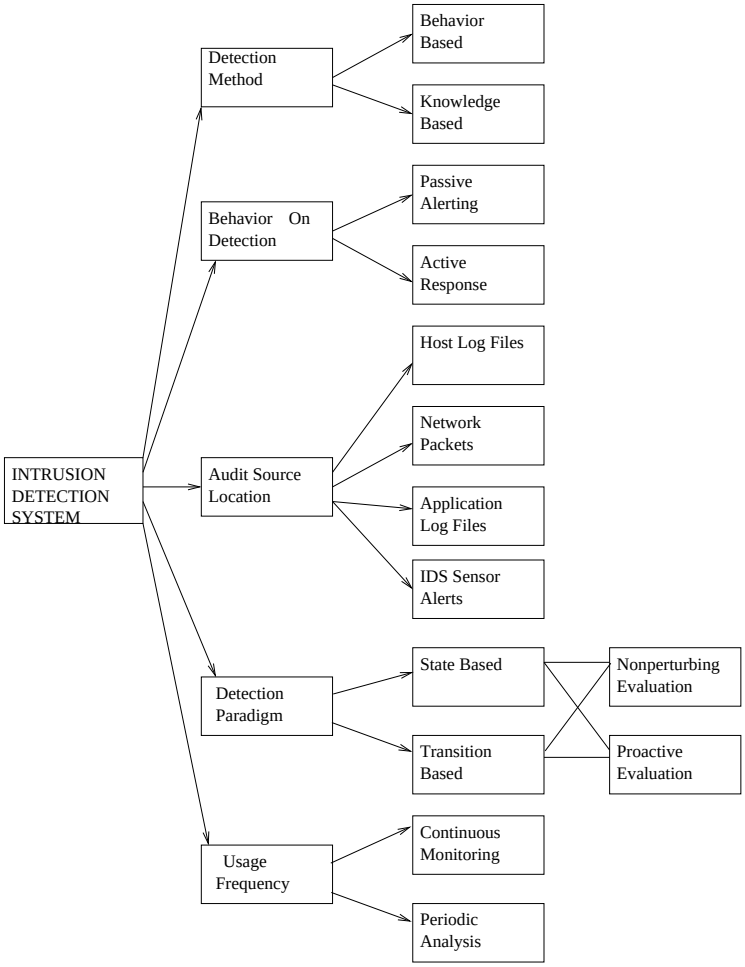

Figure 2.3: Revised taxonomy of IDS proposed by Debar et al.

cific, signature-based IDS may miss the variation on known attacks. It may also become confused when current behavior matches multiple attack signatures, either in whole or in part.

Anomaly based IDS

Also known as Heuristic or Behavior based, Anomaly based IDS analyzes the traffic patterns and determine normal activities. After that, it applies statistical or heuristic measures to the events to determine if they match with this normal behavior. Events which do not match with the accepted normal behavior patterns are considered as attacks.

By creating patterns of normal behavior, anomaly-based IDS systems can observe when current behavior deviates statistically from the normal. This capability theoretically gives anomaly-based IDSs abilities to detect new attacks that haven't been seen before or close variants to previously known attacks. It means, this types of IDS may identify any possible attack.

Since normal behavior can be changed over time, this type of system requires frequent retraining of the behavior profile, lack of which results either in unavailability of the intrusion detection system or in additional false alarms. This type of systems normally

provide comparatively high false alarms as something even slightly different from normal behavior is considered as an attack. by this types of IDS. It also requires expertise to figure out what triggered an alarm.

Host based IDS(HIDS)

Host Based IDS is installed locally on host machines. It works on information collected from within an individual computer system. It utilizes information sources like operating system audit trails, C2 audit logs, and system logs. HIDS can be installed on different types of machines namely servers, workstations and notebook computers.

Host-based IDS can analyze activities on the host it monitors at a high level of detail. It can often determine which processes and/or users are involved in malicious activities. It can monitor events that are local to a host, and can detect successful or failure of attacks that cannot be seen by a network-based IDS. Host-based IDSs can use host-based encryption services to examine encrypted traffic, data, storage, and activity. They are unaffected by switched networks and independent of network topology. They can provide thorough information gathered via logs and audit; for example Kernel logs records who the user is.

Host based IDSs are harder to manage as information must be configured and managed for every host individually. Writing to logs or reporting activity requires network traffic and can decrease network performance. Host based IDSs are network blind and cannot detect a network scan or other such surveillance that targets the entire network. They might be disabled by certain denial-of-service attacks, since these attacks may prevent any traffic from reaching the host where they are running or prevent reporting on such attacks to a console elsewhere on a network. Moreover, a host based IDS does consume processing time, storage, memory, and other resources on the hosts where such systems operate.

Network based IDS(NIDS)

Network based IDS monitors the traffic on its entire network segment. This is generally accomplished by placing the network interface card in promiscuous mode to capture all network traffic that crosses its network segment. These network traffic packets are checked by the IDS to find the attacks. Network based IDS can reassemble packets, look at headers, determine if there are any predefined patterns or signature match. Depending on this pattern or signature matching, IDS decides about the attacks.

Network-based IDSs can monitor an entire, large network with only a few well-situated nodes or devices and impose little overhead on a network. This type of IDSs are mostly passive devices that monitor ongoing network activity without adding significant overhead or interfering with network operation. They can monitor network for malicious activity on known ports such as http port 80. They are easy to secure against attack and may even be undetectable to attackers; they also require little effort to install and use on existing networks. Furthermore, they are operating systems independent.

Network-based IDSs may not be able to monitor and analyze all traffic on large, busy networks and may therefore overlook attacks launched during peak traffic periods. Packets might also be lost on flooded networks. Network-based IDSs may not be able to monitor switch-based (high-speed) networks effectively, either. Typically, network-based IDSs can-

not analyze encrypted data, nor do they report whether or not attempted attacks succeed or fail. Thus, network-based IDSs require a certain amount of active, manual involvement from network administrators to measure the effects of reported attacks.

2.2.3 Use of Artificial Intelligence in IDS

As Jeremy [21] says, issues relevant to intrusion detection include data collection, data reduction, behavior classification, reporting and response. Because of the huge amount of collected data like audit data or network traffic data, it is impossible to process them by hand. Even if we use computer, it still takes lots of time to analyze those huge data. This is why data reduction is required for further processing like classification etc. Focusing on data reduction and classification, it is found that Artificial Intelligence techniques have been used in many intrusion detections system for performing these tasks. Artificial Intelligence is nothing but a technique of solving problems as a human being does. And we know that normally humans perform a task from learning i.e. from examples and training.

There are so many AI techniques that have been used for solving intrusion detection tasks. Some mentionable AI techniques that have been used so far are: Expert systems, Rule based induction, Classifier systems like Neural Networks and Decision tree, Feature selection, Clustering etc.

Expert system solves problems by using the computer model of expert human reasoning and it requires continuous maintenance and upgrading for performing well. Behavior classification of intrusion detection systems can be done using expert systems techniques by encoding the policy statements i.e. security policy and known attacks as well as system vulnerabilities as a fixed set of rules. User behavior that matches those rules indicates that an attack in under way. However, the rules must be modified by hand.

Unlike expert systems, Rule based induction derives rules that explain the set of instances describing the problems and steps of solutions. The system acquires knowledge from these rules to solve the problem. In an IDS, rule based systems create and manage rules corresponding to anomalous behavior.

Generally, Classifier systems classify different types of patterns from a set of patterns. A classifier like Neural Network uses a model of biological system to perform classification. Another type of classifier called Decision Tree tries to separate the data into two or more groups. Then it tries to separate these groups into further groups and so on until a small groups of examples are left. More about Neural Networks will be given in next section(section 2.3). All these classifiers can be used to classify misuse or anomaly in an IDS.

Feature selection is accomplished by searching subsets of features, or information sources, and testing the ability of those features to perform the intended task. It normally reduces the amount of information required for a particular task. For instance, some data may hinder the classification process, which can be eliminated by feature selection. Moreover, some features may be redundant as the information belongs to those features is already held by other features. Thus, feature selection reduces the computation time eliminating the extra features holding the same information.

In data clustering, data are grouped together according to some common characteristics or criteria. It is used to find the hidden pattern in data that might be missed. Data

clustering techniques are also used for reducing the amount of data by dealing with the characteristics of the clusters instead of the actual data.

2.3 Artificial Neural Networks

Artificial neural network is a network of many simple processing units, each possibly having a small amount of local memory. These units are connected by some sorts of connections which usually carry numeric data, encoded by any of various means. It somewhat resembles the way human brain works. Although, in principle, neural network can compute any computable function, in practice, they are especially useful for classification and function approximation/mapping problems which are tolerant of some imprecision, which have lots of training data available, but to which hard and fast rules cannot easily be applied. The field of artificial neural networks can also be recognized using many names, such as connectionism, neuro-computing, natural intelligence systems, parallel distributed processing etc.

2.3.1 Neural Networks in General

It has always been attempted by the researchers for last couple of decades to invent a system that can learn like people. For instance, how can we make a child to know about small objects like tables, chairs etc? We just show him these objects and tell him the name of them, once, twice or as long as he is not be able to recognize them. Now we expect that he will be able to recognize those objects without any trouble i.e. we don't have to help him to recognize those objects any more. This is the idea behind how artificial neural networks work. Artificial neural networks also work in the same way. It requires training first, which can be performed through some learning process, and then it works independently.

A number of artificial neurons are normally connected together to form the artificial neural network. Neurons can be considered as small processing units and they are the basic element of the neural network. The neurons receive information by their incoming connection and transport the processed information to other neurons through their outgoing connections. These connections are called weights. The numerical information in the artificial neural networks is simulated with specific values stored in those weights. The connection structure of the networks can be changed by simply changing these weight values.

The neurons are first grouped in layers in a neural network. Each neuron in every layer, except in the input and the output layers of the net, are connected with all neurons of both the preceding and following layers. For input layer they are only connected with neurons of following layer and for output layer they are only connected with the preceding layer. The information propagates from input layer to output layer through intermediate layers called *hidden layers*. Backward journey of information i.e. from output to input layer, is also possible for some particular neural networks. There are normally no fixed number of hidden layers in a net, it can be one or more. However, some types of neural networks don't have any hidden layer.

As described in the previous chapter, Artificial Neural Network is a system loosely modeled on the human brain. It resembles the brain in two respects: one is that the knowledge is acquired by the network through a learning process and the other is that the inter-

neuron connection known as synaptic weights are used to store this knowledge. Later, this knowledge is used to perform the intended operation of the neural network. From the viewpoint of architecture, human brain consists of a specific type of cell called neuron and similarly, artificial neural network also consists of small processing unit called artificial neuron. Brief description of the natural as well as artificial neuron is given below.

The Natural Neuron

The most basic element of the human brain is a specific type of cell called neuron. Mainly the brain is composed of about 10 billion neurons and each of the neurons is connected to about ten thousand other neurons. The power of the brain comes from the number of these basic components and the multiple connections between them. There are four basic components (see figure 2.4)[1] in all natural neurons: dendrites, soma, axon, and synapses. Each neuron receives electro-chemical inputs from other neurons at the dendrites.These inputs come from the nerves or other neurons. These signals first come at the synapses which determines to what extent the signals are transmitted to the dendrites and hence, the signal is amplified by the synapses in some way. This may result in excitation, a positive contribution to the dendrite's signal, or inhibition, a negative contribution. The dendrites then transport the signals to the soma where they are accumulated. If the sum of these electrical inputs is sufficiently powerful to activate the neuron, an output spike (i.e. neuron fires at this stage) is produced and is transmitted along the axon to other neurons whose dendrites are connected to any of the axon terminals. Then these attached neurons may also fire. A neuron only fires if the total signal received exceeds a certain level.

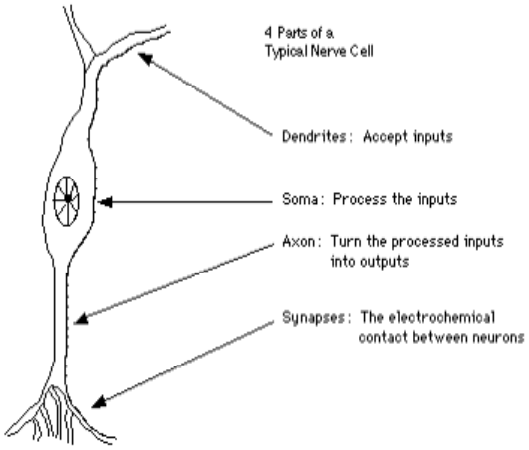

Figure 2.4: A Biological Neuron

The learning process of brain is accomplished by modifying the synaptic connections by growing new dendrites and lengthening the axon.

[1] Figure is taken from http://hem.hj.se/~de96klda/NeuralNetworks.htm

The Artificial Neuron

Like the biological neuron is the basic element of the human brain, artificial neuron is the basic elements of artificial neural networks. Artificial neuron simulates the basic functions of the natural neurons. In a biological neuron, dendrites receives electrical signal from the axons of other neurons, these electrical signals are represented by numeric values in artificial neurons. In an artificial neuron(see figure 2.5), dendrites are replaced by the input lines that receive inputs from other neurons or the input file. With each of these lines a weight is associated which acts like a synapse. Input signals are multiplied by the weights. The soma is modeled by a unit consisting of a summation followed by a thresholding function. This thresholding function is a step function albeit modern neural networks usually use the sigmoidal functions. As soon as the sum of the input signals reaches the threshold, the neuron fires, i.e. an output signal is given on the output line which is connected to other units.

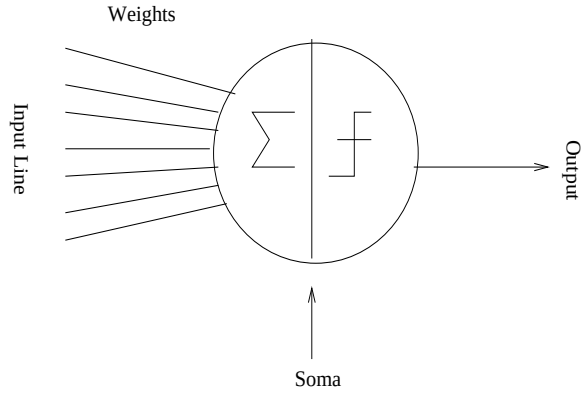

Figure 2.5: An Artificial Neuron

2.3.2 Learning Methods

The knowledge of a neural network is stored in the synapses, which are the weights of the connections between the neurons[26]. These weights between two layers of neuron can be represented as matrices. This knowledge represented as weight matrix is used to calculate the correct output for the particular input to the network. And this knowledge can be acquired by the learning process called 'training'. Normally training is performed by presenting the input pattern associations to the network in sequence. The weights are adjusted to capture the knowledge from the input pattern and stored. This process is continued as many times as it requires to produce satisfactory results. Once the learning process is finished, i.e. the network can present the correct output pattern from the given input pattern with current weight, the current weights are stored and used for intended future operation. There are mainly two types of learning methods used for training a neural network: *Supervised* learning method and *Unsupervised* learning method.

Supervised Learning

In this specific type of learning method, the output pattern of the neural net is compared with a target output pattern corresponding to the input pattern. Depending on the difference between the output and target patterns, the net error is computed. Weight adjustments are done taking these errors into account. Thus, the learning in this method are benefited from the assistance of the teacher. This type of learning method is used in the system where the output patterns are specific for corresponding input patterns. Which means we know the output pattern and we force the system to learn about that particular output pattern for corresponding input pattern. Figure 2.6 shows a schematic diagram of supervised learning. Many types of neural networks e.g. *Perceptron, Backpropagation Neural Networks* etc(*cf. sec.* 2.3.4) used supervised learning methods for their training.

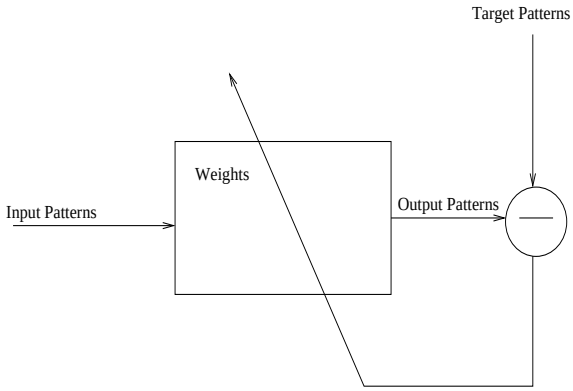

Figure 2.6: A schematic diagram of Supervised Learning.

Unsupervised Learning

Unlike the supervised method, in unsupervised learning method, no target patterns exist. In general, an unsupervised learning method is one in which weight adjustments are not made based on comparison with some target output. Instead, the network learns to adapt based on the experiences collected through the previous training patterns. And hence, the learning is performed without the benefit of any teacher. This types of learning methods are normally used for performing clustering of patterns. The way how the patterns will be grouped is normally defined explicitly or implicitly in the learning algorithm itself. Figure 2.7 shows a schematic diagram of unsupervised learning. Examples of neural networks that used unsupervised learning method for their training are *Hopfield Net, Kohonen Feature Map* etc(*cf. sec.* 2.3.4) .

2.3.3 Connection Structures

The connection structure of neural net deals with how the neurons are connected between layers. There are normally two types of connection structures: *Feedforward* and *Feedback*.

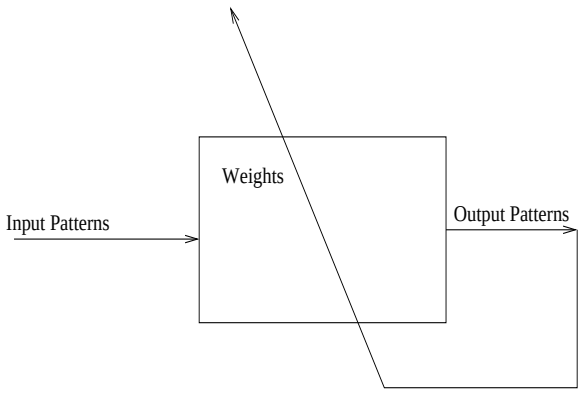

Figure 2.7: A schematic diagram of Unsupervised Learning.

Feedforward

In this specific connection structure of a neural net, neurons of one neuron layer may only have connections to neurons of other layers. An example of such a net type is the *Perceptron*(*cf. sec.* 2.3.4). Figure 2.8 shows a neural network with feedforward connection structures.

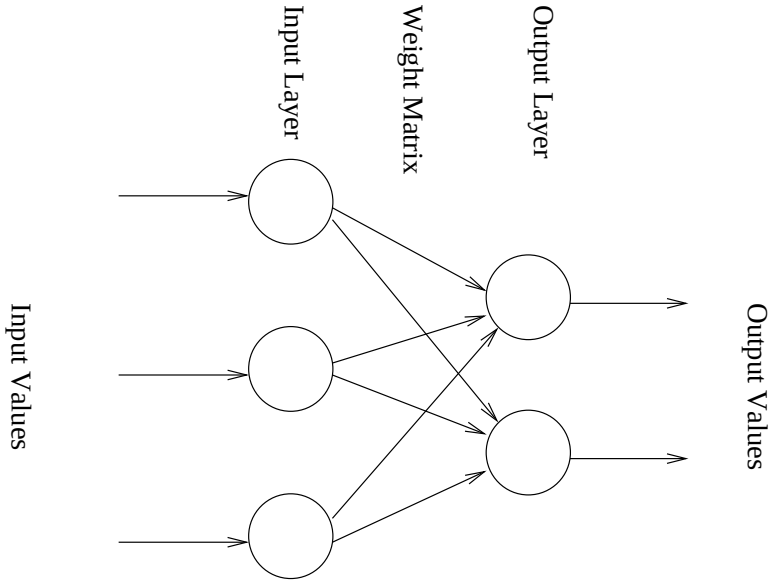

Figure 2.8: Perceptron: An example of Feedforward structure.

Feedback

This is a type of connection structure of a neural net, where neurons of one neuron layer may have connections to neurons of other layers and also to neurons of the same layer. An example of such a net type is the *Hopfield Net*(*cf. sec.* 2.3.4). Figure 2.9 shows a neural network with feedback connection structures.

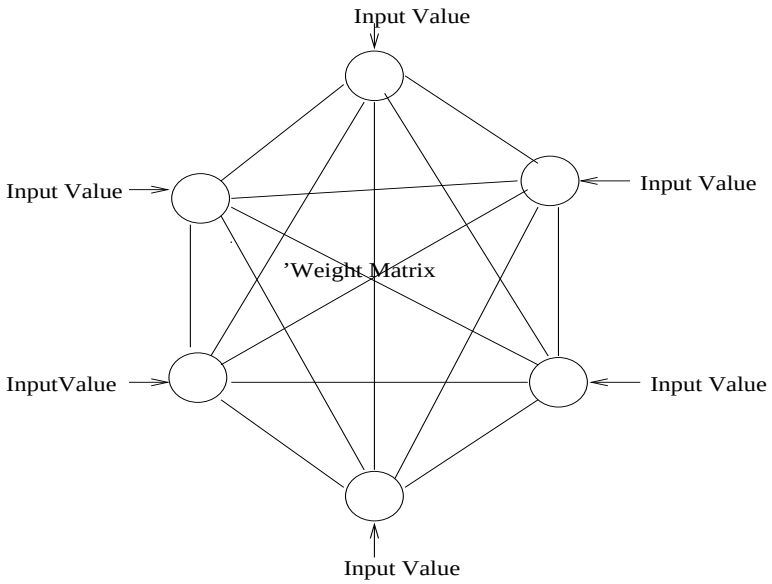

Figure 2.9: Hopfield net: An example of Feedback structure.

2.3.4 Types of Neural Networks

There are many types of neural networks available. Actually the number is not limited. Researchers are inventing new types of neural networks(or at least the variations of the old ones) almost in every week [9]. The types of neural networks can be distinguished by their physical structure, learning method and connection structures etc. A taxonomy of neural networks followed by a selection of neural networks is given below. A cumulative representation of the characteristics of different neural networks is given in the Table 2.1.

Taxonomy of Neural Networks

The field of artificial neural network is evolving so fast that it is difficult to find a concrete taxonomy to classify different types of neural networks. Although new types of neural networks are invented almost every week, Paplinski [30] has given a taxonomy that begins with two phases of neural networks, active or decoding phase and learning or encoding phase. Artificial neural networks can be classified into *feedforward*(static) and *feedback*(dynamic,recurrent) systems from the viewpoint of their active phase, and *supervised* and *unsupervised* systems from the viewpoint of their learning phase.(see figure 2.10 for the taxonomy).

Perceptron

The perceptron, an invention of F.Rosenblatt in 1958, was one of the earliest neural network models. This is a very simple neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. *Bias* is a constant value(generally 1.0) used as a single input to the network. During training, when all the input values in the network are equivalent to zero, 'bias' helps to update the weight values. Perceptron has only two layers: input and output layer. It does not have any hidden layer. It is a feedforward neural networks that learns with a

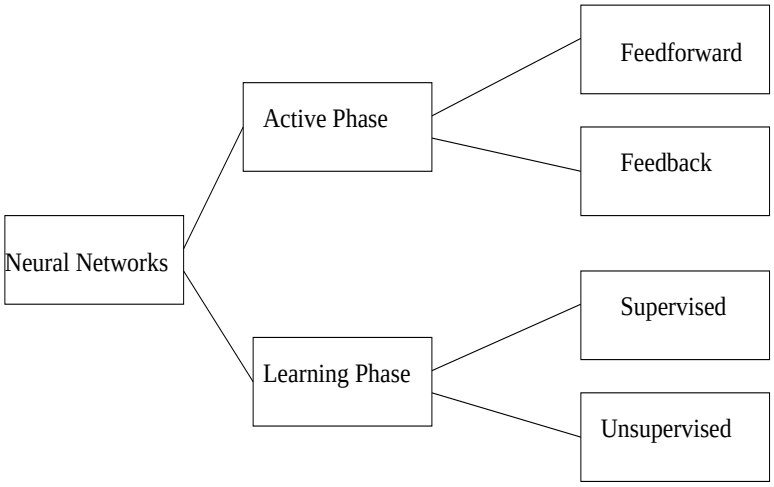

Figure 2.10: A taxonomy of Neural Networks

supervised learning method. Perceptrons has several limitations. First of all, the output values of a perceptron can take only one of two values(true or false). Secondly, they can only classify linearly separable sets of vectors. A set of input vectors are linearly separable if a straight line or plane can be drawn to separate the input vectors into their correct categories. The problem of boolean AND and boolean OR functions are linearly separable but boolean XOR function is not linearly separable and hence, percrptron cannot solve the XOR problem. Figure 2.11 shows how we can separate the illustration of AND and OR function with a single line and how we cannot do the same for XOR function.

Perceptrons are mainly used in simple logical operations and pattern classification.

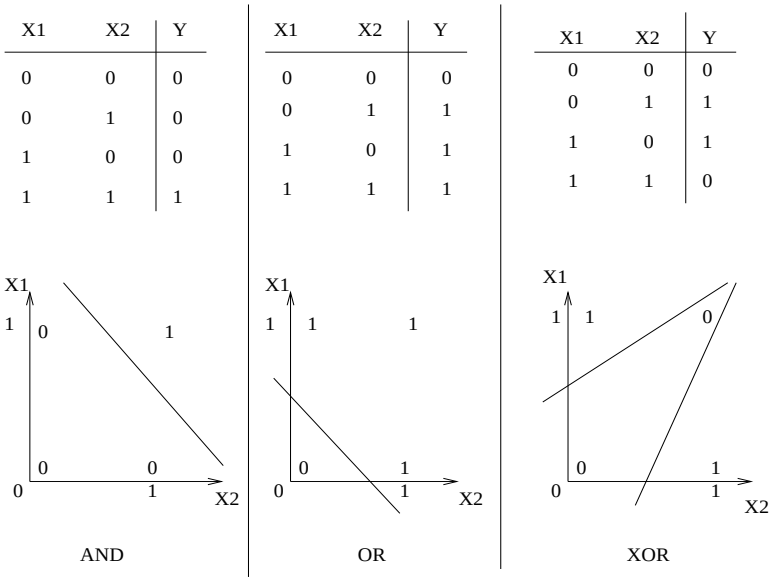

Figure 2.11: Boolean functions AND and OR are linearly separable, XOR is not

Multilayer Perceptron(MLP)

This is the most common neural network model [37]. It is an extension of perceptron with one or more hidden layers between its input and output layers. With its extended structure it has more capability and thus, it can solve more complex problems including boolean XOR function. This types of neural networks are used for comparatively complex logical operations and pattern classification.

Backpropagation Net

Backpropagation neural network is one of the most powerful neural networks [31]. It has the same structure as multilayer perceptron and mainly used in complex logical operations, pattern classification and speech analysis.

Like in multilayer perceptron, backpropagation neural network has three layers: input, output and hidden layers. Before the training of the net, i.e. any data has been run through the network, weights are simply random numerical values, much like a newborn's brain, developed but without knowledge. When presented with an input pattern, each input node takes the value of the corresponding attribute in the input pattern. Then, each node in the hidden layer multiplies each attribute value by a weight and adds them together. If this is above the node's threshold value, it fires a value of '1', otherwise it fires a value of '0'. The same process is repeated in the output layer with the value from the hidden layer, and if the threshold value is acquired for this layer, it represents the classified pattern for the corresponding input pattern, which is compared with the actual classification pattern and error value is calculated. This error value is then backpropagated through the network, and the weights of output and hidden layers are adjusted with these error values. This process is repeatedly carried out until it satisfies a predefined termination condition. The net is then assumed 'trained' and the weights are stored. However, the weight change is performed in such a way that the current point on the error surface will descend into a valley of the error surface, in a direction that corresponds to the steepest(downhill) *gradient* or slope at the current point on the error surface [4]. For this reason, backpropagation is also called *gradient descent method*.

The advantages of backpropagation neural nets are that they are great at prediction and classification. On the other hand, there is always a lack of explanation of what the net has learned. Moreover, although reasonable, they are slow compared to other learning algorithms [32].

Hopfield Net

Hopefield nets are principally used for auto-association. If a distorted input vector is given, the Hopfield Net associates it with an undistorted pattern stored in the network. Hopfield net consists of a set of neurons and all of them are connected each other by a unique weight. It means, it has only one layer, there is no difference between input and output layers.(see figure 2.9) The main applications of a Hopefield Net is storage and recognition of patterns e.g. image files.

Kohonen Feature Map

Kohonen Feature Map, also known as Kohonen's Self-Organizing map, is a type of unsupervised learning. It has two layers namely one input layer and one feature map. The feature map is a neuron layer where neurons get organized according to certain input values. The connection structure of Kohonen Feature Map consists of both the feedforward and feedback i.e. neurons are connected from input layer to feature map as well as each other within the feature map.

2.4 Other Works on IDS and Artificial Neural Networks

So far, a number of works have been accomplished for developing intrusion detection systems for both misuse and anomaly detection by using artificial neural networks. In the context of this thesis work, a brief description of some of the research on anomaly detection using neural network has been given below. The description contains research works on both host and network based anomaly detection.

2.4.1 Neural Network Intrusion Detector(NNID)

This is a neural network based anomaly intrusion detection system in a host computer proposed and developed by Jake Ryan et.al [12]. The system is based on identifying a legitimate user based on the distribution of 'commands' she or he executes. In this system, the audit logs for each user for a period of several days has been obtained and used as training data. For each day and user, a vector has been formed that represents how often the user executed each command. Based on these commands distributor vectors, the neural networks has been trained to identify the user. The network is then presented with each new command distribution vector and if the network's suggestion is different from the actual user, or if the network does not have a clear suggestion, it signals an anomaly.

Data is collected on a machine for 12 days, resulting on 89 user-days as the machine has total 10 users and all of them are not regular users. A set of 100 most common commands in the logs has been used as input. The number of times a command is used is divided into intervals. There are total 11 intervals and they are represented by values from 0.0 to 1.0 in 0.1 increments. The first interval meant the command has never been used; the second that it has been used once or twice, and so on until the last interval where the command has been used for more than 500 times. These values, one for each command, are then concatenated into a 100-dimensional command distribution vector to be used as input to the neural network. The standard three-layer backpropagation architecture has been chosen for the neural network. The input layer consisted of 100 units, representing the user vector; the hidden layer had 30 units and the output layer 10 units, one for each user.

The network has been trained on 8 randomly chosen days of data(65 user vectors) and the remaining 4 days of data has been used for testing(24 vectors). The system is reported to obtain an accuracy of 96% in detecting unusual activity, with 7% false alarm rate.

2.4.2 Intrusion Detection Using Neural Networks and Support Vector Machines

In this work [18], S. Mukkarnala et al. demonstrates a neural network based system for anomaly detection. This neural network based system has been developed for detecting intruders in computer network.

The system consists of three phases. First, The raw TCP/IP dump data has been processed into machine readable form using automated parsers. Secondly, the neural network is trained on different types of attacks and normal data. The input has 41 features and the output assumes one of two values: intrusion(22 different attack types) and normal data. And lastly, the system performance has been tested on the test set containing 6980 data. All Training and test data have been acquired from the 1998 DARPA [36] intrusion detection evaluation program.

The training of the neural networks was conducted using feed forward back propagation algorithm. Three different neural networks with following architectures have been used:

1. Network A : 4-layer, 41-20-20-20-1.
2. Network B : 3-layer, 41-40-40-1.
3. Network C : 3-layer, 41-25-20-1.

The network is set to train until the desired mean square of training error of 0.001 is met. A data set of 7312 normalized input-output pairs consisting attack patterns, and normal user patterns has been used for training. As mentioned above, a data set of 6980 test patterns has been used for testing the system performance.

The performance of detection by three different architectures is claimed as : Network A performed with an accuracy of 99.05%; network B achieved an accuracy of 99.25% ; network C performed with an accuracy of 99%.

2.4.3 Network-Based Intrusion Detection Using Neural Networks

This anomaly detection system that detects network-based attacks is proposed by Alan Bivens et al. [19]. In this system, MLP neural networks has been used to examine traffic data. Moreover, Self-organizing map(SOM), a kohonen neural network has been used for grouping the network traffic data together to present it to the neural net.

This system is a modular network-based intrusion detection system that analyzes tcpdump data to develop windowed traffic intensity trends. In the learning approach, some of the components need time to be trained on the traffic intensity before detection is possible, this phase is called architectural learning period as during this time machine ports are selected for monitoring, the structure of neural network is determined, SOM is trained and the neural network is trained. Once these steps have been taken, the system can stop the learning phase and begin detection.

In detection phase, the system reads in network tcpdump data and sends it first to the preprocessing module which keeps statistics of the traffic intensity on a source by source basis in each time interval. At any given point, there can be many sources in communication with the victim. Therefore, simply grouping the information by sources will not

create a uniform representation of data for the neural network. To address this issue, the preprocessed source information, which contains the traffic intensity from a source to the target machine, is send to a clustering module which groups sources with similar traffic intensity trends together during the training phase. As mentioned above, clustering is accomplished by the Self-organization map(SOM). The number of thus created clusters is constant and established in the architectural learning phase. When a source is assigned to a particular cluster, the source's traffic intensity is simply combined with the traffic intensity of that cluster. This is done at each time interval, allowing a source to be assigned to different clusters in different time intervals. Once the traffic intensity is grouped into clusters, the group totals are first sent to the normalization module for formatting and then to the actual neural network to render a decision as the likelihood of a pending attack.

The system is tested with tcpdump binaries from the DARPA 1999 training dataset. It has been claimed that the system performs well when tested on individual types of attacks one at a time rather than detect all types of attacks simultaneously.

For union off all attacks, the system gives 100% of normal predictions with 0% false negatives and only 24% correct attack prediction with 76% false positives.

On the other hand, for a particular type of attack called *sshprocesstable*, the system gives 100% correct predictions for both normal and attacks with 0% false negative and false positive. However, it has been declared that the training and testing in this area was limited due to the lack of many instances of the same attack in the available dataset.

2.4.4 A Neural Network Based Intelligent Intrusion Detection System

This is a rather recent work on intrusion detection. Robert Birkely [38] has proposed and designed a neural network based network intrusion detection system in his masters thesis in June 2003. This is an anomaly based intrusion detection system.

In this system, a backpropagation neural network has been used to classify users' behavior and detect the intruders therefrom. Users' behavior is classified from the tcpdump data of a network and attacks are detected from this classification. The neural network has 132 input nodes in the input layer, 12 nodes in the hidden layer(there is only one hidden layer) and two nodes in the output layer. The input patterns for the neural net have been chosen from the sniffed network traffic of a simulated computer network. Each session is considered as one input pattern. Sessions are presented in text form in the available network traffic. They are converted into binary form with small modification and data reduction. Hence, 132 binary bits are acquired for each single session which eventually used as the input pattern for the neural net. In the output layer, two nodes are used to represent either attack or normal. These nodes with the shape 1-0 is considered as attack and 0-1 is considered as normal. Training and test data has been collected from DARPA intrusion detection evaluation data sets [35] . The network has been trained with different number of epochs for observing the improvements.

This system is trained and tested only with a small amount of training and test data. Only 197 pairs of input-output patterns, of which 98 sessions with normal traffic and 97 sessions with attack, have been used for training the network. In three test approaches, 50, 40 and 100 sessions are used respectively for testing the system performance. It has been reported that the system gains 86% accuracy of anomaly detection with 0% false

positives.

2.5 Summary

Signature based IDS keeps a database of signatures of known attacks. For detecting intruders, it finds the match with those signatures. Anomaly based IDS classifies known users' behavior first and something that does not match with this behavior is considered as intrusion. Host based IDS detects intruders in a single machine and Network based IDS detects intruders in the whole network.

Artificial intelligence technique like artificial neural network can be used for classification purpose in an intrusion detection system. Neural networks learn by training. Two types of learning methods are used to train a neural network : supervised, where the net is trained by both the input and output pattern; and unsupervised, where the net is trained only by the input pattern. A neural network has one input layer, one output layer, and zero or more number of hidden layers. All these layers contain a number of neurons, the basic element of neural networks. All neurons in different layers are connected each other in two ways: feedback and feedforward. Neural networks are classified from their internal structure as well as learning methods used for training. Table 2.1 shows a number of neural networks and their characteristics.

Other artificial intelligence techniques used for intrusion detection are rule-based systems, expert systems, data clustering, feature selection etc.

Neural Networks	**Learning Method**	**Connection Type**	**No. of Layers**
Perceptron	Supervised	Feedforward	2
Multilayer Perceptron	Supervised	Feedforward	3 or More
Backpropagation	Supervised	Feedforward	3 or More
Hopfield Net	Unsupervised	Feedback	1 matrix
Kohonen Feature Map	Unsupervised	Feedforward/ Feedback	2, 1 input, 1 map

Table 2.1: Different types of Neural Networks and their characteristics

Chapter 3

Design

3.1 Introduction

As discussed in chapter 2, intrusion detection can be classified according to two design criteria: it can be anomaly(behavior) based or signature based, depending on how intrusions are detected and it can be host based or network based, depending on where intrusions are detected. The objective of this thesis work is to develop an Anomaly Based Network Intrusion Detection System. In an anomaly based system, the main task is to recognize and distinguish behavior of different authorized system users and identify intruders from that knowledge. Behavior of a particular user can be formed from his[1] habit of using computer. For instance, a particular user may use only some specific types of protocols, or he may only use some particular machines i.e. particular IP addresses, or he may only work within a specific time range e.g. during office hours, and so on. This information can be obtained from the network traffic. Thus, it is possible to store normal users' behavior and intruders can be recognized from the distortion of normal behavior. In that case, system administrator can be notified to take necessary action.

3.1.1 Design Goals

We are to design an intrusion detection system which is supposed to be:

- Offline system, in the sense that the time difference between the presence of intruders in the network and detection can be arbitrarily long. However, we aim to ensure that this delay will be very small, if not almost instantly.
- Efficient, so that it can handle large volume of training and test data in a reasonable amount of time.
- Flexible, with the presence of a controller that will control the frequency of alarming for false positives.
- Automatic, so that it can handle and classify all user information without human involvement.

[1] 'he' is considered as 'he or she' throughout the document

- Portable, so that the system can be used for different types of networks.
- Cost effective by using a small amount of memory to store knowledge about normal user behavior.
- Powerful, so that it can accommodate large number of users' behavior from a large computer network.

3.1.2 Overview

Our objective is to design a behavior based system to detect intruders in a computer network. Operation of the system is divided into three phases:

1. Input Data Collection and Preprocessing
2. Training
3. Detection.

In preprocessing phase, network traffic is collected and processed for use as input to the system. In the second phase, this system gathers knowledge about the normal behavior of the network users from the preprocessed input data, and store the acquired knowledge. In the detection phase, the system detects attacks based on the knowledge which is achieved during the training phase, and notify the system administrator. A block diagram of system overview is given in figure 3.1.

As mentioned earlier, the main task is to generalize and classify user behavior and detect intruders from this classification. This task may be carried out using various approaches like expert systems, rule based induction, artificial neural network approaches etc. We have selected neural networks to accomplish our task. This is because, they have some preferable properties that make them suitable for the task of classification.

Some of the preferable properties of artificial neural networks are:

- It is their inherent property that they can learn through training.
- Their complex internal structures enable them to learn and accommodate large number of patterns.
- They can generalize the knowledge acquired through training for similar patterns.
- They have efficient storage capability of a large set of patterns.

Because of these attractive characteristics of neural network, we have chosen it for classifying user behavior in our system. Since, in the training phase of our classification task, both the input and output patterns are available, we can use a supervised learning method i.e. we can take the help of a teacher to train the neural net. Hence, we have decided to use backpropagation neural network in this regard.

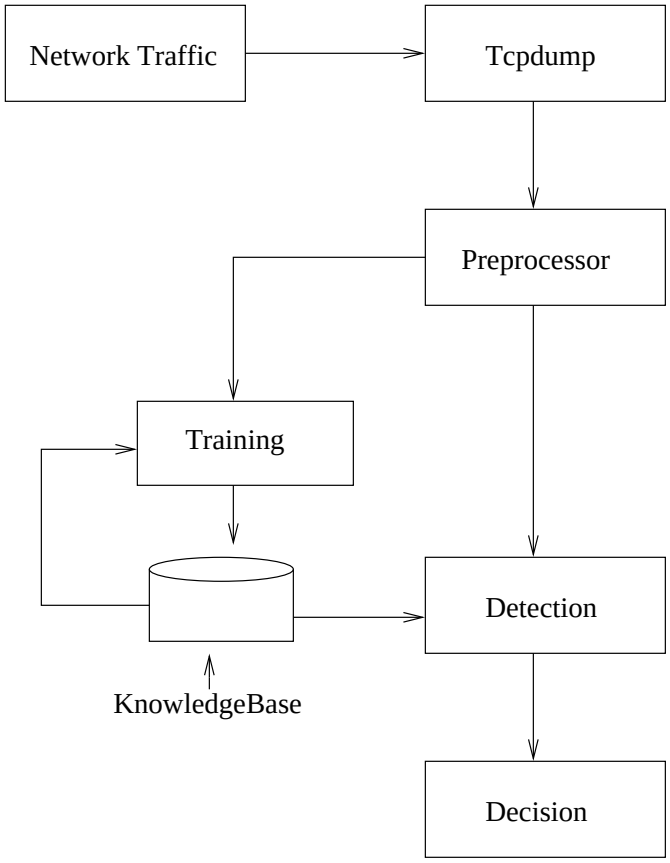

Figure 3.1: Block diagram of system overview

3.2 Collecting and Preprocessing Input data

Network traffic can be collected either from a real computer network or from a simulation of network. Due to the unavailability of a suitable real computer network, we propose to use network traffic data called DARPA intrusion Detection Evaluation, collected by the Information Systems Technology Group(IST) of MIT Lincoln Laboratory from a simulated network [35].

Each session from the network traffic is used as an input pattern of our system. These traffic sessions are represented in text form in our collected data. To suit the input format of the backpropagation neural network, we convert these traffic patterns into binary form. Moreover, in some cases, we require some manipulation to remove redundant information or to compress huge amount of data into a smaller amount.

Figure 3.2 shows a sample representation of 16 network sessions. Each session consists of following pieces of information separated by space: *Session serial number, Start date, Start time, Duration, Session server name, Source port, Destination port, Source IP address, Destination IP address, Attack score and Attack name.* Among them, Session serial number, Start Date and Session server name have to be removed. This is because, Session serial number is irrelevant as it has nothing to do with a user's behavior; Start Date is unnecessary as for every single day in a year, there is a unique Start date; session server name is also represented by the destination port number, so it is redundant. Moreover, Attack score and name are simply replaced by a single code which only indicates the presence of

an attack.

```
1 01/23/1998 16:56:12 00:01:26 telnet 1754 23 192.168.1.30 192.168.0.20 0 −
2 01/23/1998 16:56:15 00:00:13 ftp 1755 21 192.168.1.30 192.168.0.20 0 −
3 01/23/1998 16:56:17 00:00:01 smtp 43493 25 192.168.0.40 192.168.1.30 0 −
4 01/23/1998 16:56:17 00:00:00 auth 1756 113 192.168.1.30 192.168.0.40 0 −
5 01/23/1998 16:56:19 00:00:01 smtp 43494 25 192.168.0.40 192.168.1.30 0 −
6 01/23/1998 16:56:19 00:00:00 auth 1761 113 192.168.1.30 192.168.0.40 0 −
7 01/23/1998 16:56:19 00:00:01 ftp−data 20 1762 192.168.0.20 192.168.1.30 0 −
8 01/23/1998 16:56:22 00:00:00 ftp−data 20 1767 192.168.0.20 192.168.1.30 0 −
9 01/23/1998 16:56:24 00:00:02 ftp−data 20 1768 192.168.0.20 192.168.1.30 0 −
10 01/23/1998 16:56:25 00:01:01 telnet 1769 23 192.168.1.30 192.168.0.20 0 −
11 01/23/1998 16:56:27 00:00:00 ftp−data 20 1770 192.168.0.20 192.168.1.30 0 −
12 01/23/1998 16:56:36 00:00:03 finger 1772 79 192.168.1.30 192.168.0.20 0 −
13 01/23/1998 16:56:42 00:00:03 smtp 1778 25 192.168.1.30 192.168.0.20 0 −
14 01/23/1998 16:56:43 00:00:03 smtp 1783 25 192.168.1.30 192.168.0.20 0 −
15 01/23/1998 16:56:45 00:00:00 http 1784 80 192.168.1.30 192.168.0.40 1 phf
16 01/23/1998 16:56:49 00:00:14 ftp 43504 21 192.168.0.40 192.168.1.30 0 −
```

Figure 3.2: Sample sessions from network traffic

After removing this information from the network traffic, the new format only includes: Start time, Duration, Source Port, Destination port, Source IP Address, Destination IP address, and Attack code(see the conversion in figure 3.3). Among them, start time is modified into different time slots and rest are converted into binary form.

Start time shows the time when the particular session starts. Users' habit of using specific types of protocol in some specific time can be determined from this piece of information. For instance, a user may check his email before starting his daily work. But this does not mean that he checks his mail exactly at the same time for every single day. This time might vary for a couple of minutes. This is why we have decided to make some different time slots so that every start time falls into a specific time slot. We make each time slot of 15 minutes duration. The duration could be 10 minutes or 30 minutes or whatever. But a time slot of 10 minutes seems too small and 30 minutes seems comparatively bigger.

The following is the detail description of the conversion and modification:

Start time: This piece of information is converted into different time slots of a day. Each slot consists of fifteen minutes, it means there are four slots in an hour and hence, there are 96 time slots altogether for 24 hours. Instead of representing the specific start time of the session , we represent the binary representation of the time slot number where it belongs to. For instance, 08:11 belongs to time slot number 33 (i.e. (8X4) + 1, as '11 minute' shows that it belongs to the first quarter of an hour). Seven binary bits are enough to represent upto decimal 96.

Duration: It shows the total time duration of the corresponding session and can be represented by 18 binary bits- 6 for hours, 6 for minutes and 6 for seconds; since in case of hours, the maximum number is 23 and in case of minutes as well as seconds, the maximum number is 59, they can be represented by minimum 6 binary digits.

Source and Destination Ports: The source and Destination port numbers range from decimal 0 to 65535 which can be represented by 16 binary digits.

Source and Destination IP Addresses: The actual size of IP address in *IPv4* is 32 binary bit. These addresses are normally represented as four parts, where each part is represented by 3 decimal digits. It means each part of an IP address needs 8 binary bits to be converted into binary form. Here, IP addresses are represented in decimal form. So, they have to be converted back into binary form which consists of 32 binary bit.

Attack Code: The state of attack can be either 'yes' or 'no'. It means, an attack can simply be represented by a '1' and normal traffic can be represented by a '0', hence, it requires only one binary digit.

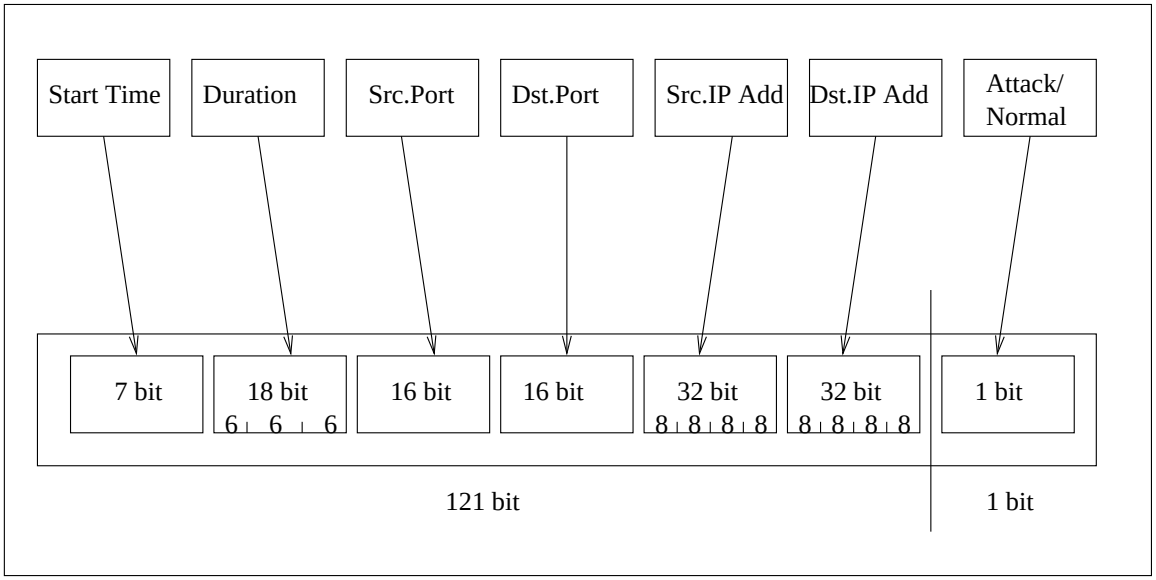

Figure 3.3: Converting a session into binary form

These seven types of information of each session can be divided into two groups. First six pieces of information can be concatenated and represented as a single line of binary bits to form the first group which will be considered as the input pattern to the system. And the single bit of attack state can be considered as the second group which will be used as corresponding output pattern. We have, in this stage, kept both of the groups together in a single line of binary bits keeping in mind that we will feed them separately to the system for carrying on the next phase of operation. All these preprocessed sessions can be stored in one or more files.

3.3 Training

In this phase, user behavior is classified from the network traffic sessions. As mentioned earlier, a backpropagation neural network is used for this purpose. The input to the neural network is the first group of preprocessed traffic session and the output of the neural network is the second group i.e. either attack or normal.

In the training phase, we work in two different steps. First of all we retrieve sessions from the file and separate them into two groups: one is for input and the other is for

target output. And secondly, we train the network with the input data, check if the generated output pattern satisfies the target output pattern and calculate the error from the distortion of target output. We retrain or stop training the network depending on this error value. Once the training is over, we store the knowledge it acquired.

3.3.1 Network Architecture

The input pattern made from each session consists of total 121 binary bits. Hence, we have total 122 input nodes in the input layer, where 121 is for the input and the other one is for the bias. As we have mentioned in the description of *Perceptron* in sec. 2.3.4, bias is a constant value which is used as a single input to the network. During training, 'bias' helps to update the weight values when all the input values in the network are equivalent to zero. We have only one output unit in the output layer. The only node in the output layer represents binary 1 or 0, where 1 means attack and 0 means normal(see figure 3.4). Moreover, we have only one hidden layer and it consists of 40 nodes. The reason behind choosing 40 nodes for the hidden layer is that we need some 'brain' for our network as it has to deal with a large amount of data. If the network has too few hidden units in the hidden layer, the neural net will suffer from the lack of brain and the quality of a prediction will drop. On the other hand, if it has too many units in the hidden layer, it will work very well on the familiar data, but will fail on the data that was never presented before [33].

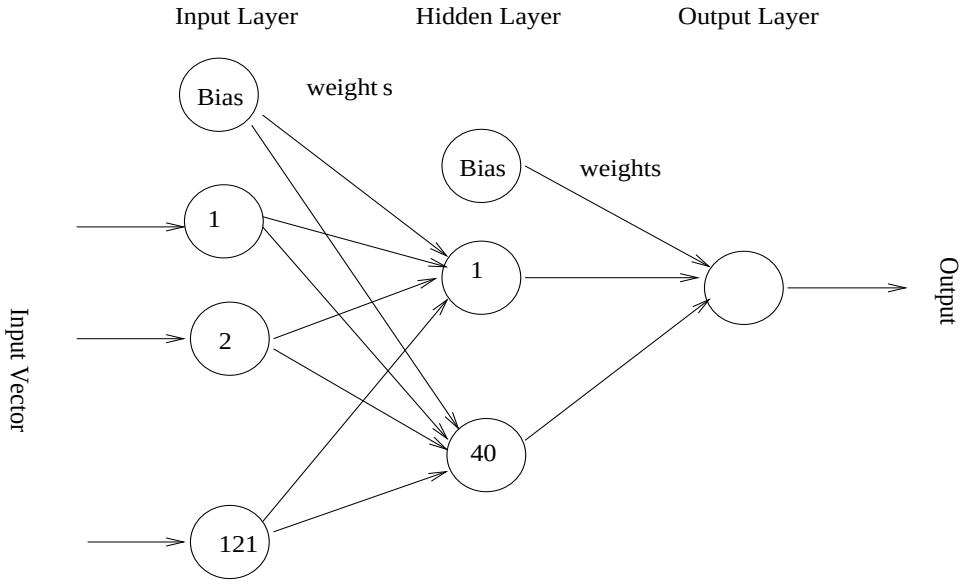

Figure 3.4: Neural Network Architecture

3.3.2 Retrieving input data from file

Our preprocessed data is stored in the files and it has to be retrieved and fed to the system. One file contains data for all sessions of one single day. We retrieve all the data of one external file and store it in a temporary file. We take each single session at a time from this temporary file and feed the network. Each session is nothing more than a vector consisting both the input and output patterns. All the sessions of current file can be fed

to the network using an inner loop. The whole process can be repeated for all the external files using an outer loop.

3.3.3 Training the network

In the beginning of the training, the weight values of the backpropagation neural network have to be initialized. For the first time of training, they are initialized with random values from -1 to +1. This is because, if all weights in the network are initially set to the same value and the solution to the problem requires unequal weights, the network will never learn since error changes are proportional to the weight values [8]. For consecutive training periods, they are just initialized with the previous stored values.

The input vector of the network is 122 bits. Among them first bit is the bias which is always '1'. The other 121 bits are taken from the first 121 bits of the session vector that has been retrieved from the file. As stated above, the 122th i.e. last bit of session vector is used as the target output.

There are many patterns available for training the network. We train the network with one pattern at a time by calculating the error value and updating the weights. Training the network with all available input patterns defines an epoch. We continue the training for a specific number of epochs depending on the termination condition. The termination condition might be a reasonably predefined number of epochs or until an inferior error value is achieved.

When the training is terminated , the updated weight values are stored in external files. These weight values can be used for further training or for classification.

3.4 Detection

In this phase, the system is already trained with the normal behavior of user and from that knowledge it will detect the intruders or abnormal behavior of authorized users. This step is to be done similarly to the training step with some reservation. As described in training step (*sec. 3.3*), we retrieve sessions from the file and get the input pattern for the network. In this phase, the input pattern is same as training but there is no output pattern. This is because, the network itself predicts the possible output pattern for each given input pattern.

We have the same network architecture in this step like training. We retrieve input data from file in the same way as in training step except that we discard the output pattern i.e. the last bit of session vector, as there is no use of output pattern in this case.

Like we have trained the network, in the beginning of detection, the network has to be initialized with the weights. In this step, we initialize the network with the weight values that we have stored after training. This is the knowledge of the network that has to be used to find the abnormal behavior. After that, we feed the input vector to the network and get the output produced by the network. This output will show us whether the given input pattern represents normal behavior or an attack. Unlike training phase, we do not calculate any error value in the detection phase. The same process is continued for all the sessions of all the test files and only for one single epoch.

Chapter 4

Implementation

4.1 Introduction

In this chapter, we will discuss the implementation issues related to our solution. We describe the main algorithms as well as the data structures used in our implementation. As we have discussed in the design chapter, the discussion will be done in the following phases:

- Input Data Preprocessing.
- Training.
- Detection.

We implement these three phases in three different modules. Three different Java classes have been coded for this purpose. They are: *preProcess.java*, *TrainNet.java* and *ClassNet.java*.

4.2 Input Data Preprocessing

As mentioned in section 3.2, our system takes input data from a simulated network. Each data file contains a number of sessions from network traffic data in a text form. As we know from previous chapter, each session consists of following pieces of information: *Session serial number, Start date, Start time, Duration, Session server name, Source port, Destination port, Source IP address, Destination IP address, Attack score and Attack name*. On the file, these pieces of information are separated by a blank space except that the Session serial number is separated by two consecutive blank spaces. Each session is separated from other session by a new line.

Our system retrieves one character at a time from the file and store it in a temporary array. Whenever a blank space is retrieved in this way, it assumes that one complete piece of information of the corresponding session is retrieved and stored in the temporary array. For instance, first two spaces indicate that the stored information is session serial number(this is because, session serial number is separated from other information with two consecutive spaces), third space indicates the start date, fourth indicates the start time

and so on, so forth. In this stage, the system takes necessary action like data reduction, conversion etc. on the retrieved information. According to our system design, our system discards the information of serial number, start date, session server name and attack name. The start time is converted into time slot and in binary form. Other pieces of information are converted into binary form as well. The processed data is stored in the output file accordingly.

As all the information is represented in decimal form and has to be converted into binary form, we use the following conversion algorithm for converting from decimal into binary:

Algorithm ConvertIntoBinary

```
// num is the given decimal number
// Array temp holds the converted binary number
  j=0;
  while ( num > 0 )
  begin
     temp[j] := ( num mod 2 )
     num     := ( num div 2 )
     j++
  end

  for( k = j-1 down to 0)
  begin
     return temp[k]
  end
```

Here, the decimal number is repeatedly divided by 2 until it becomes 0 and in every repetition the remainder is taken as the leftmost bit of the binary number.

Time Slot conversion

As we get the *start time* in a text form, first we convert it into corresponding decimal number. Start time has three different parts: hour, minute and second. We convert only hour and minute into decimal. As second is a very small amount of time, we simply ignore it. The conversion process is very easy, since hour and minute are represented by only two decimal digits, we multiply the rightmost bit with 1 and the leftmost bit with 10 and add them. And thus, we get the equivalent decimal value of the corresponding text representation. In the second step, we convert this decimal number into corresponding time slot. As we have discussed in the previous chapter, each time slot consists of a quarter of an hour. Following algorithm is used to make the time slot:

Algorithm CalculateTimeSlot

```
// Let min is given minutes
// Let minSlot represents calculated minute slot for the given minutes
// Let hrs is the given hours
```

```
// Let num is the desired time slot

  if( min < 15 ) then
      minSlot := 1
  endif
  if(( min >= 15 ) AND ( min < 30 )) then
      minSlot := 2
  endif
  if(( min >= 30 ) AND ( min < 45 )) then
      minSlot := 3
  endif
  if( min >= 45 ) then
      minSlot := 4
  endif

  num := ( hrs * 4 ) + minSlot

  return num
```

The time slot can be calculated in more than one ways, for instance, we can simply add the outcome of division of the minutes by 15 and the multiplication of the hour by four. However, we use the above mentioned one in our system.

After getting the time slot in decimal form, we simply convert it into binary form and store on the output file.

Time Duration conversion

Time duration has three parts: hour, minute and second. We process them separately. For each part, we first convert them from text to decimal and then from decimal to binary. At the end, they are stored on the output file.

Port Conversion

First, the text representation of the port number is converted into decimal form. In order to do this, we find the total number of decimal digits that represent the port number. This can easily be found from the total length of the array holding the port number. Now we use the following algorithm to find the corresponding decimal number:

```
// Let length is the total number of decimal digits
// Let portNum holds the text representation of  port number

   multiple := 1
   for (i= length -1 down to 0)
   begin
      num := num + portNum[i] * multiple
```

```
    multiple := multiple * 10;
end
return num
```

IP Address Conversion

IP address has four parts. Like we have done before, we take each part separately, convert them from text to decimal and then from decimal to binary. Finally, the binary values are merged with the output file.

Output Pattern

The output pattern is represented by one single binary bit. We check whether the session contains an attack or not. If there is no attack in the session data the output pattern is stored as a binary '0', otherwise it is a binary '1'. We simply ignore the attack name, if any.

4.3 Training

We have already mentioned in the previous chapter that we use a backpropagation neural network for behavior classification. We construct a neural network with 121 units in the input layer and one unit in the output layer. We have only one hidden layer and we have the option to choose different number of hidden nodes for our system. This can simply be done by changing the value of the variable for hidden units in our implementation. In addition to the input and hidden layer, there are one extra unit in the input layer and one extra unit in the hidden layer. These two units called 'bias' are permanently connected to constant values(generally 1.0) and they are used for internal thresholding purpose of the neural network. The inputs to the neural network are the input pattern consists of 121 binary bits. Each of these bits is the input to one unit in the input layer of the neural network. The first 7 units in the input layer get inputs from the time slot of a session. Similarly, next 18 units get inputs from the time duration of a session and so on. For each pair of input units and hidden units there is a connection between them. Similarly, each hidden layer is connected to the output unit as we have only one output unit. The bias in the input layer is connected to each of the units in the hidden layer except the hidden bias. Similarly, hidden bias is connected to the output unit. Figure 4.1 shows how the biases as well as the input vector connected to the input layer and the output produced by the network. A weight-value is associated with each of the connections. All weight values are randomly initialized to values in a pre-defined range(in our implementation, from -1.0 to +1.0).

The only output unit in the output layer correspond normal or attack i.e. 0 or 1, represented as real number. This unit is designed in such a way that when the input pattern is a normal, the corresponding output obtains a value close to 0.0. On the other hand, if the input pattern represent an attack the corresponding output obtains a value close to 1.0.

The neural network is implemented using a collection of data structures and algorithms. Each unit in the input, hidden and output layers can be thought of as tiny processing

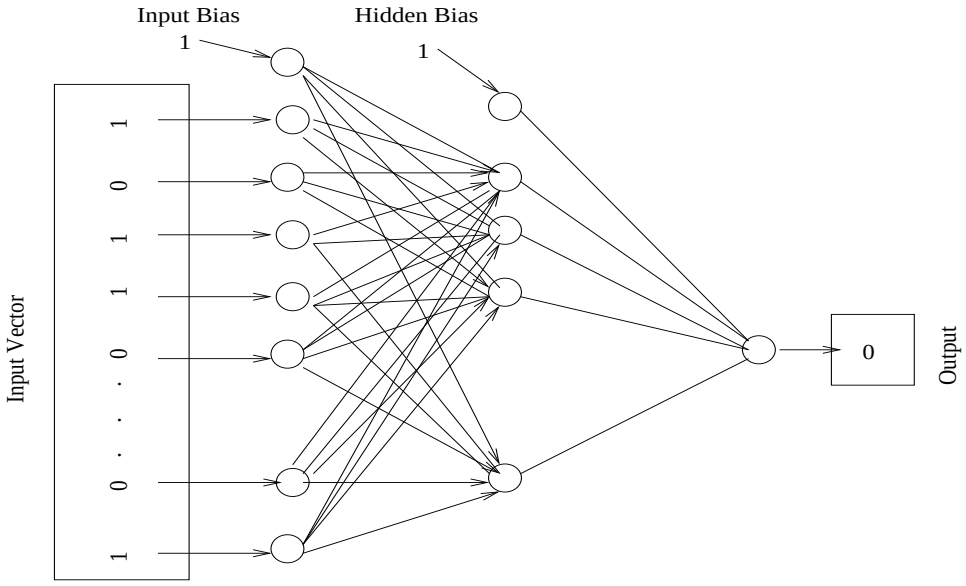

Figure 4.1: Input vector, Bias, output and the neural net

elements. These elements have one or more inputs, one or more outputs and some limited processing capabilities. The data structure to represent such a unit can be shown as follows:

```
Structure neural_net
  {

      inputs : array of real numbers
      output : real number

  }
```

4.3.1 Retrieving input data from file

For retrieving input data from files, we define an array containing all the input file names. Here, in this step, by input file we always mean the file that has been created after processing the raw data. According to our need for the training purpose, we may retrieve data from all the files or from some particular files using a loop. We can always change the parameter of the loop by hand. However, each file contains many lines of data, and each line has the input output pair for the training or testing of neural net. Each line consists of 122 digits, among them, first 121 digits are for input pattern and the last digit is the corresponding output pattern. For training, both input and output are used but for testing, only input pattern is used.

4.3.2 Backpropagation Algorithm

We have briefly discussed about backpropagation neural network in section 2.3.4. Following is the presentation of the algorithm of the backpropagation neural network [27, 28, 29]:

Algorithm Backpropagation

```
// Input units,  X[i], i = 1,2,3...., numInput   ; X[0] is the bias
// Hidden units, H[i], i= 1,2,3...., numHidden   ; H[0] is the bias
// Output units, O[i], i= 1,2,....., numOutput
// Weights between input and hidden layers W1[i,j],
                                          i=0,1..,numInput;
                                          j=0,1..,numHidden
//  Weights between hidden and output layers W2[i,j],
                                          i=0,1..,numHidden;
                                          j=0,1..,numOutput

1. Initialize, W1[i,j] = random(-0.1, 0.1)
                                       for all i=0,..numInput,
                                              j=0...numHidden
   Initialize, W2[i,j] = random(-0.1, 0.1)
                                       for all i=0,..numHidden,
                                              j=0...numOutput
2. X[0] = 1.0    // Input Bias
   H[0] = 1.0    // Hidden Bias

3. For each of the input-output pairs do the following:

   a) Assign activation levels to the input units
                                    (from the input vector)
   b) Calculate,

        H[j] = 1/(1+exp(-sum(W1[i,j].X[i]))) ;
                                      for all j=1,...,numHidden
                                              i=1,...,numInput


   c)  Calculate,

        O[j] = 1/(1+exp(-sum(W2[i,j].H[i]))) ;
                                      for all j=1,...,numOutput
                                              i=1,...,numHidden

   d) Compute the errors of the units in the output layer denoted del_2[j],

     del_2[j] = O[j](1-O[j])(T[j]-O[j])  for all j=1,...,numOutput.
       where, T[j] is the corresponding target output.

   e)  Compute the errors of the units in the hidden layer denoted del_1[j],

     del_1[j] = H[j](1-H[j]) sum(del_2[i].W2[j,i])
                                       for all j=1,...,numHidden.
                                              i=1,...,numOutput
```

```
f)  Adjust the weights between the hidden and output layer,

   W2[i,j] = W2[i,j]+ ETA.del_2[j].H[i] ; for all i=0,..,numHidden,
                                                  j=0,..,numOutput
    where, ETA is the rate of learning.

g)   Adjust the weights between the input and hidden layer,

   W1[i,j] = W1[i,j]+ ETA.del_1[j].X[i] ; for all i=0,..,numInput,
                                                  j=0,..,numHidden
```

4.3.3 Training the network

For training the neural network, our first step is to initialize the weight values. Then we calculate the outputs of the hidden neuron from the inputs as well as from the weight values. We then calculate the error from the predicted output and the target output. In the next step we update weights. When training is over, we store the latest weights.

Weight Initialization

We implement two methods for weight initialization: *initWeights()* and *initPrevWeights()*. The former one is used in the very beginning of training. It initializes the weight values with random values between -1.0 and 1.0. The later method is used when we retrain the network. It simply initiates the weights with the latest stored weights from the weight files.

Calculate the output of hidden layer and output layer

We implement this step in the method *calcNet()*. In this step, outputs of the units in the input layer are propagated to the associated connections where they are multiplied by corresponding weights. These weight values are then propagated to the inputs in the units of the hidden layer. The output of each of these hidden units is calculated from these inputs based on the formula stated in the step 3(b) of algorithm of backpropagation(sec. 4.3.2) :

$$H_j = \frac{1}{1 + e^{-\sum_{i=0}^{numInput}(W1_{i,j}.X_i)}}$$

These outputs are again propagated to the connections between hidden and output layers and the weighted values are input to the output unit. Again using the formula from the step 3(c) of the backpropagation algorithm(sec. 4.3.2), output of this output unit is calculated:

$$O_j = \frac{1}{1 + e^{-\sum_{i=0}^{numHidden}(W2_{i,j}.H_i)}}$$

This propagation of values from input units to the output unit is called ***forward propagation***.

Calculate error values and update Weights

The calculation of backpropagation error and updating the weights are implemented by the method *weightChangesHO()* and *weightChangeIH()*. In the training phase, an output pattern is provided for each of the input patterns. Once predicted output is available in the output layer of the neural network, this predicted output is matched with the supplied target output and error is calculated. Errors for the output unit are calculated as follows:

```
del_2[j]=O[j](1-O[j])(T[j]-O[j])  for all j=1,...,numOutput.
            where, T[j] is the corresponding target output.
```

These errors are back propagated to the hidden units and errors for each hidden units are calculated as follows:

```
del_1[j]=H[j](1-H[j])(del_2.W2[j])  for all j=1,...,numHidden.
```

This later formula is a bit different than we described in the algorithm. This is because, we have only one output unit in the output layer.

Based on these errors, all connections of the neural network are updated. The process of updating each weight during learning in backpropagation algorithm is called *delta rule* [4]. However, the weights are updated as follows:

For weights between hidden and output layer:

```
W2[i] = W2[i,j]+ ETA.del_2.H[i] ; for all i=0,..,numHidden
```

For weights between input and hidden layer:

```
W1[i,j] = W1[i,j]+ ETA.del_1[j].X[i] ; for all i=0,..,121,
                                                  j=0,..,numHidden
```

Here, 'ETA' is a factor affecting the rate of learning of the neural network. The largest possible learning rate that does not cause *oscillation* should be selected. As Lou [8] says, an example of oscillation can be imagined where a network's position is halfway down a valley on a two dimensional error surface. If the learning rate is too large, the network's next step might be to the other side of the valley, as opposed to moving closer toward the bottom. Then the following step might return to the original side. In this example, the network would not be making any progress toward the bottom where the solution lies.

Conversely, if the learning rate is too small, training could take too long to get to the bottom of the valley. However, we have chosen 0.1 as learning rate in our implementation.

In this way, errors in the output layer are propagated in the backward direction and weights of the connections are corrected. This propagation of errors is termed as ***backward propagation*** or simply ***backpropagation***.

Store Weights

This is the last step of training phase. When the training is over, we store the latest updated weights on some weight files. We might have stored these weights in each and every epoch but that would take more time for training. This is why, we plan to store only the latest weight values. All weights between input and hidden layer are stored using method *storeWeightIH()* on file named *weightIH* and weights between hidden and output layer are stored using method *strore WeighHO()* on file named *weightHO*. The idea is very simple, we take each weight value and store it on corresponding output file in a new line.

4.4 Detection

As we have mentioned before, neural network works in two modes: Training and Detection. We have already discussed about the training of neural network in the previous section. The detection phase is carried out in almost same way as training. In section 4.3, we mentioned that the training of backpropagation neural network consists of two phases: forward propagation and backpropagation. In detection, only forward propagation is required.

4.4.1 Forward Propagation Algorithm

The algorithm for forward propagation is same as backpropagation(*cf.sec. 4.3.2*). Only exception is that in step 1, we initialize weight values with the predefined weights from file. Moreover, the backpropagation part i.e. step 3(d,e,f,g) is absent in the forward propagation. Following is the complete algorithm:

Algorithm Backpropagation

```
// Input units,  X[i], i = 1,2,3...., numInput   ; X[0] is the bias
// Hidden units, H[i], i= 1,2,3...., numHidden  ; H[0] is the bias
// Output units, O[i], i= 1,2,....., numOutput
// Weights between input and hidden layers W1[i,j],   i=0,1..,numInput;
                                                      j=0,1..,numHidden
//  Weights between hidden and output layers W2[i,j], i=0,1..,numHidden;
                                                      j=0,1..,numOutput

1. Initialize W1[i,j] from the weight file weightsIH
   Initialize W2[i,j] from the weight file weightsHO

2. X[0] = 1.0     // Input Bias
```

```
   H[0] = 1.0     // Hidden Bias

3. For each of the input-output pairs do the following:

   a) Assign activation levels to the input units(from the input vector)
   b) Calculate,

         H[j] = 1/(1+exp(-sum(W1[i,j].X[i]))) ; for all j=1,...,numHidden
                                                       i=1,...,numInput


   c)  Calculate,

         O[j] = 1/(1+exp(-sum(W2[i,j].H[i]))) ; for all j=1,...,numOutput
                                                       i=1,...,numHidden
```

4.4.2 Detection and Response

In detection phase, we use stored weight values from the training phase to initialize the weight vectors. We directly fetch the weight values from external file and carry out our task of initialization. Hence, method *initWeights()* of detection works in the same way as method *initPrevWeights()* of training.

After initialization of the weights, the input units of input layer are activated with the input patterns that has been taken from the input file. The outputs of the input layer are propagated towards the output layer similarly as training. The calculated output from the output unit of output layer is considered as the desired output pattern. If this output pattern is close to 1.0, it is considered that the input pattern contains an attack. We use a threshold variable in our implementation to determine the threshold value for detecting an attack. This threshold value works like a controller. For example, if we set this value 0.75 and if the predicted value of output pattern becomes more than 0.75 then corresponding input pattern will be considered as an attack. And thus, this threshold value has a role to control the frequency of detecting false positives. If this value is too high(close to 1.0), the system will be very strict on detection and hence, there is a possibility of having less false alarms. But it may miss to detect some attacks because of the restriction. On the other hand, if it is too low(close to 0.0), the system will have a tendency of detecting more false positives. We keep this value 0.5 in our system but we can always change this threshold value by hand.

Our input patterns are represented in binary form. So, we implement a method *binToDec()* for converting this binary information into text representation of decimal equivalent. This is because, when an attack is detected, we use this method to show the detail of the corresponding input pattern in decimal text form, so that the system administrator can understand very easily and can take necessary action promptly.

4.5 Management of Files in Our System

We have three different kinds of text files. Files containing raw tcpdump data which have been downloaded from the web site of the Lincoln Laboratory of MIT [35] can be stored in any directory. These files are not used directly to our system. By preprocessing, these raw files are converted into input files which contain input vectors for our system. These later types of files are to be kept in the current directory along with the Java classes for training and detection. Java Class *TrainNet.java* generates the weight files after training and store them in the current directory. Accordingly, java class *ClassNet.java* retrieved weights from the weight files which are kept in the current directory. The files that have to be checked for attacks are also kept in the current directory.

Chapter 5

Evaluation

5.1 Introduction

This chapter highlights on experimental issues. First of all we discuss on suitable training and test data sets. These data sets are used to experiment our system. We train our system in three different approaches. In the first approach, the system is trained with all network traffic generated in consecutive three weeks. In the second approach, the system is trained with network traffic of one selected week. And in the last approach, the system is trained with randomly selected small amount of network traffic. Different test data sets are also used for testing the system performances for all these approaches. A discussion on the results obtained and the comparison of achieved results with other works are given. The possibility of future extension from the performed experiments is also mentioned. This chapter ends with a summary.

5.2 Training and Test data

Before going to experiment our system, we need to decide on the training as well as test data which is used for experiments. We have three sets of training data for three different approaches. As mentioned before, we have taken our data from DARPA intrusion evaluation data sets.

5.2.1 DARPA Data Sets

The Information Systems Technology Group(IST) of MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency(DARPA ITO) and Air Force Research Laboratory(AFRL/SNHS) sponsorship, has collected and distributed the first standard corpora for evaluation of computer network intrusion detection systems [35]. Such evaluation efforts have been carried out in 1998 and 1999. We have decided to use data sets of 1998 DARPA Intrusion Detection Evaluation. These data sets consist of seven weeks of network based attacks in the midst of normal background data. We download all the data sets of seven weeks from the Lincoln Laboratory website [36] for further processing for our system. We make our own data sets from these data of seven weeks for training and testing our system in various ways.

5.2.2 Training Data Sets

As we have mentioned, we have total three approaches to train our neural network. These approaches are based on training the network with different training data sets. Therefore, three different data sets are used for training these three different approaches.

First Approach

For the first approach, we take data of first three weeks from the DARPA data sets. These data of three weeks are stored in three different files. We process these huge data for our system and again store them in three different files. It is our system that retrieves this data automatically from all different files.

Second Approach

In the second approach, we still use huge data but less than that of the first approach. We use data of only one week for training the network. In this case, we select all data from the fourth week.

Third Approach

In this approach, we use relatively very small amount of data. We select only 450 sessions of normal data i.e that does not contain any attack, randomly from the network traffic of seven weeks. Moreover, all network traffic of seven weeks contain total 166 attacks. We divide this data into two groups. One group contains 350 normal and 100 attacks, which is used for train the network. The other group of data consists of rest 100 normal and 66 attacks, which is eventually used as unknown normal and attacks for testing the system performance for this approach. However, in the training phase, the neural network never see the data of this second group and this is how it becomes unknown to the system.

5.2.3 Test Data Sets

We decide to test our system for all three approaches in four different ways. Each approach will be tested with:

1. known normal data.
2. unknown normal data.
3. known attacks.
4. unknown attacks.

Here, known normal and attacks mean, the data that has been used to train the network. Unknown normal and attacks mean the normal data and attacks that has neither been used for training nor been seen by the network before.

Therefore, we have four sets of test data for each of the approaches. However, we have two groups of data for the third approach. One group consists of relatively small amount of data and the other group contains all the sessions of network traffic of one full working day. As mentioned above, each group contains four sets of test data.

The numbers of input patterns for each set of each approach are as follows:

First Approach

For this approach, each data set contains following amount of input pattern:

- Known normal: 500
- Unknown normal: 200
- Known attacks: 35
- Unknown attacks: 70

Known normal and attacks are selected randomly from the data sets of first three weeks. This is because, the net, in this case, has been trained with the data from first three weeks, which means the system already knew about this data. As the net is trained with huge data, we use more known normal data for this test data set. Moreover, we have total 35 attacks in first three-week data. Unknown normal and attaks are selected randomly from the rest four weeks of data. Due to the availability of more unknown attacks, we use 70 unknown attacks for this data set. In this case, we consider that more number of unknown attacks will increase the rate of perfection for the system performance.

Second Approach

For this approach, each data set contains following amount of input patterns:

- Known normal: 500
- Unknown normal: 500
- Known attacks: 24
- Unknown attacks: 141

For this approach, the net is trained only with the data sets of fourth week which contain total 24 attack sessions. All these 24 attacks are taken as known attacks. Known normal data is selected randomly from the data sets of fourth week. Unknown data(both normal and attack) is selected randomly from the data sets of other weeks. As more unknown attacks are available, we use 141 unknown attacks for this data set.

Third Approach

As we mentioned, we have two different groups of data sets for this approach. For the first group of data sets, we have same amount of data for each set. Each set of this group has 50 input patterns.

For the second group, we take all network traffic from Monday of first week. It contains total 33856 input patterns. Among them, only two patterns contain attacks, rest of them contain normal data.

5.3 Training and Experimental Results

The system has been implemented, trained and experimented in the Linux environment(Red Hat Linux version 7.2) on IBM compatible PC(Intel Celeron 2000 MHz Processor). In this section, we train and experiment our system using different data sets. According to the three different training approaches, we divide our experiment into three categories. Details experimental results of different approaches are given in this section. Some tabular representations of results are also given for having quick impression about the developments. Moreover, the list of parameters used to train the neural network is given in table 5.1.

5.3.1 Termination Conditions

We have mentioned several times in the previous chapters that the training of neural network is continued until a termination condition is satisfied. Generally, the training terminates when the root mean square(RMS) of the error value in the output layer of neural net is reduced to an acceptable level. The lower the RMS error, the better the training. However, too low RMS error may over-train the network. In this case, the neural net detects only those patterns that are identical to the patterns used for training. It means, the neural network looses its prediction capability. Another termination condition is the number of epochs. In that case, the neural net is trained for a predefined number of epochs.

We train our network for different number of epochs for different approaches. The actual number of epochs is determined prior to each training approaches. However, we have also a RMS error value as a termination condition. This is because, the neural network might be over-trained with the specified number of epochs if the RMS error becomes too low. In this case, the training is stopped earlier if the specified RMS error value is already obtained. Our specified RMS error value is 0.001.

Parameter	Value
Input Units	121
Output Units	1
Hidden Units	40
Learning Rate	0.1
RMS error	0.001

Table 5.1: Parameters of neural network

5.3.2 First Approach

In this approach, we train the network with the above specified data, i.e. data from first three weeks. We have altogether 732656 patterns in first three weeks. After 100, 200, and 500 epochs of training, we test the system performance with above mentioned test data sets.

After 100 epochs, the system can recognize 500 out of 500 known normal data, 200 out of 200 unknown normal data, 22 out of 35 known attacks and 30 out of 70 unknown attacks. It means, the detection rate for normal data(both known and unknown) is 100% and for known and unknown attacks are 63% and 43% respectively. After 200 epochs, the results remain almost unchanged. The system can recognize 500 out of 500 known normal data, 200 out of 200 unknown normal data, 23 out of 35 known attacks and 28 out of 70 unknown attacks. It means, the detection rate for normal data(both known and unknown) is 100% and for known and unknown attacks are 66% and 40% respectively. The detection rate of known attacks has slightly improved while the detection rate of unknown attacks has deteriorated.

After 500 epochs, detection rate of normal data remains same. We can see a slight improvement of the detection for both known and unknown attacks. The system can recognize 500 out of 500 known normal data, 200 out of 200 unknown normal data, 22 out of 35 known attacks and 30 out of 70 unknown attacks. It means, the detection rate for normal data(both known and unknown) is 100% and for known and unknown attacks are 69% and 43% respectively.

Table 5.2 shows the details detection results and table 5.3 shows the corresponding detection rate.

No. of Epoch	**Known Normal**	**Unknown Normal**	**Known Attacks**	**Unknown Attacks**	**RMS Error**
100	500/500	200/200	22/35	30/70	0.00622387
200	500/500	200/200	23/35	28/70	0.00610859
500	500/500	200/200	24/35	30/70	0.00618525

Table 5.2: Experiment results for the first approach

No. of Epoch	**Known Normal**	**Unknown Normal**	**Known Attacks**	**Unknown Attacks**
100	100%	100%	63%	43%
200	100%	100%	66%	40%
500	100%	100%	69%	43%

Table 5.3: Detection rate of the first approach

It should be mentioned here that training the system with such a big amount of data requires a lot of time and the neural network converge very slowly. As our intension is to extend this thesis work to a real-time system in future, we need to keep the time consumption for training as low as possible. At this stage of training, the network is taking approximately 3 minutes for every single epoch. Moreover, we see from the RMS error that after 500 epochs the error value has been increased which is supposed to be decreased.

This is happened because of the huge training data and the larger learning rate. We can choose smaller learning rate but that will take even more time for training. Hence, we stop training the network after 500 epochs and step forward for the second approach of training.

5.3.3 Second Approach

We have experienced from the previous approach that the network consumes a lot of time to be trained for a huge data. We have one week of training data for this approach. This data set contains total 234190 training patterns. So, we can say, we still have a huge data. This is why, we plan to train the network very carefully. If it again consumes a lot of time without much improvements, we will simply step forward for the next step of experiment.

We start our first phase with 100 epochs. In the test session, system successfully detects all the normal data for both known and unknown normal. It also detects 23 known attack out of 24 and 94 unknown attacks out of 141. After trained for 200, 300 and 500 epochs respectively, the system returns the same detection rate with slight improvement of detecting unknown patterns. For all these phases, it detects 97 unknown pattern out of 141. Detail results are given in table 5.4. Regarding time consumption, in this epoch, one single epoch requires approximately 54 seconds.

No. of Epoch	**Known Normal**	**Unknown Normal**	**Known Attacks**	**Unknown Attacks**	**RMS Error**
100	500/500	500/500	23/24	94/141	0.00962212
200	500/500	500/500	23/24	97/141	0.00876911
300	500/500	500/500	23/24	97/141	0.00865387
500	500/500	500/500	23/24	97/141	0.00849281

Table 5.4: Experiment results for the second approach

Detection rate of the second approach is given in table 5.5.

No. of Epoch	**Known Normal**	**Unknown Normal**	**Known Attacks**	**Unknown Attacks**
100	100%	100%	96%	67%
200	100%	100%	96%	69%
300	100%	100%	96%	69%
500	100%	100%	96%	69%

Table 5.5: Detection rate of second approach

5.3.4 Third Approach

In this approach, we have trained the network with different number of epochs and the system has been tested with all four sets of test data. The network has been trained for 50, 100, 200, 500, 1000 and 5000 epochs respectively. The results obtained from the experiment of this approach is given in table 5.6.

No. of Epoch	Known Normal	Unknown Normal	Known Attacks	Unknown Attacks	RMS Error
50	50/50	49/50	49/50	49/50	0.06411281
100	50/50	49/50	49/50	47/50	0.05205014
200	50/50	49/50	49/50	47/50	0.04882187
500	50/50	50/50	50/50	48/50	0.00812874
1000	50/50	50/50	50/50	48/50	0.00505765
5000	50/50	50/50	50/50	48/50	0.00197661

Table 5.6: Experiment results for the third approach

There are 50 test patterns for each data sets. After trained with 50 epochs, the system detects 50 known normal data, 49 unknown normal data, 49 known attacks and 49 unknown attacks. After 100 epochs, the detection rate remains almost same except that the detection of unknown attacks is 47. After 200 epochs, the result remains same. After 500, 1000 and 5000 epochs, the system returns the same result with slight improvement of detecting unknown attacks. 48 unknown attacks are detected out of 50. Interestingly, training consumes very small amount of time. It takes only 52 seconds to train the network for 500 epochs. The detection rate of this approach is given in table 5.7.

Trained with Epoch	Known Normal	Unknown Normal	Known Attacks	Unknown Attacks
50	100%	98%	98%	98%
100	100%	98%	98%	94%
200	100%	98%	98%	94%
500	100%	100%	100%	96%
1000	100%	100%	100%	96%
5000	100%	100%	100%	96%

Table 5.7: Detection rate of third approach

Further Test

It is seen from table 5.6 that the system is working very well when we test it with small amount of data. What, if we test the system with huge data? In the next step, we test the system with above mentioned second group of data for the third approach. There are total 33856 input patterns in this data set, among them only two are attacks. Before going to test with this data set, we train the network for 500 epochs with the training data. This is because, we have got satisfactory result from the last experiment with this amount of training. However, the system recognize 121 false positives in addition to two existing attacks. It means the rate of detecting attacks is 100%, but it raise a lot of false alarms.

For solving the problem of so many false positives, we retrain the network. But this time we add these 121 normal sessions that has been detected as attacks, with the training data. After retraining for 500 epochs, we get 20 false positives. So, we retrain the network again by adding these 20 sessions with the previous data. This time the network succeeds without

raising any false alarm. Here, by retraining, we always mean a fresh training, we don't use any previous stored weights for retraining. However, in the actual implementation, we have carried out this part by hand. It means, the system does not automatically takes and merge the data with the training data set. We simply copy and paste it manually.

For the whole process of retraining, we test the system with first group of test data as well. Table 5.8 shows the results obtained from the second experiment.

Training	**No. of Epoch**	**No. of False Positives**	**Known Normal (Detection Rate%)**	**Unknown Normal (Detection Rate%)**	**Known Attacks (Detection Rate%)**	**Unknown Attacks (Detection Rate%)**
Previously Trained	500	121	50/50 (100%)	50/50 (100%)	50/50 (100%)	48/50 (96%)
Re-Trained	500	20	50/50 (100%)	48/50 (96%)	50/50 (100%)	41/50 (82%)
Re-Trained	500	00	50/50 (100%)	50/50 (100%)	50/50 (100%)	45/50 (90%)

Table 5.8: Experiment results for third approach with huge data

5.3.5 Discussion on Results

For training, Approximate time consumption by all three approaches is given in table 5.9. However, it is seen from all three approaches that, first two approaches consumes a lot of time for training. The converging rate is also very low, it means the RMS error decreases very slowly after a number of epochs of training. Although the detection rate for normal data is very promising(100% for both known and unknown normal data) for both of the approaches, the detection rate for attacks is not satisfactory.

On the other hand, third approach consumes very small amount of time for training but performs very well. It detects 100% normal data(both known and unknown), 100% known attacks and 96% unknown attacks. It means, the overall detection rate for attacks is 98% with 0% false positives.

In the second step of third approach i.e. test with huge data, system requires retraining and gives a detection rate of 100% for both known and unknown normal data, 100% for known attacks and 90% for unknown attacks.

No of Epoch	**First Approach**	**Second Approach**	**Third Approach**
500	24 hours	7.5 hours	54 seconds

Table 5.9: Approximate time taken for training the neural net

5.3.6 Comparison with Other Related Works

We have mentioned some related research works on intrusion detection using neural networks in Chapter 2. Alan Bivens et al.[19] uses both Multi-layer Perceptron(MLP) and Self-Orgaining Map(SOM) for their network anomaly detector(*cf. sec. 2.4.3*). They claimed that their system gives 100% correct normal predictions i.e. 0% false negatives and only 24% correct attack prediction. In our case, we have 98% correct attack prediction.

Robert Birkely [38] works almost the same way as we do. In his work(*cf. sec. 2.4.4*), he gets 86% detection rate on attacks. But the system is tested with only very small amount of data(less than 200 patterns). In our case, we get 98% correct prediction of attacks for small amount of test data. However, we have extended our work by testing with huge data(33856 patterns) which generates some false positives. We solve this problem by retraining technique and get a detection rate of 95% with 0% false positive.

Figure 5.1 shows a comparison chart for our work and other researches.

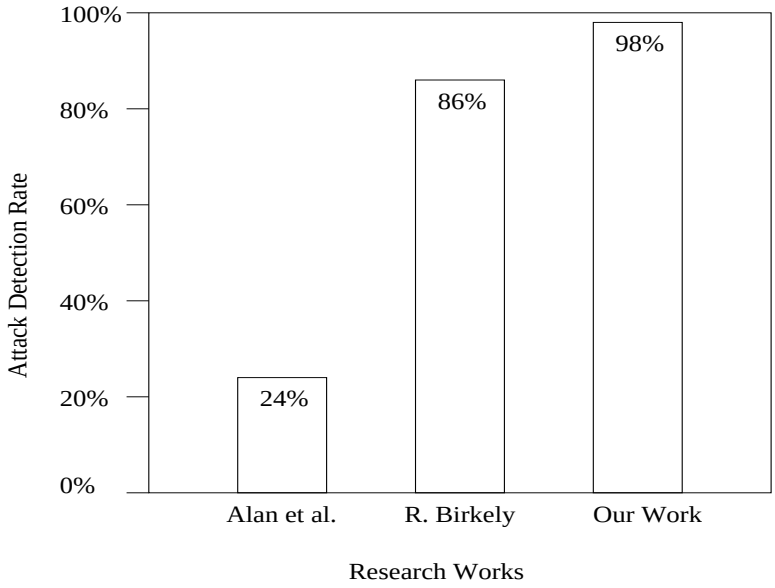

Figure 5.1: Comparison of performance of related works

5.4 Summary

The system has been trained in three different ways and tested with different sets of test data. For all three approaches, the detection rate for normal data(both known and unknown) is 100%. It means we have 0% false positive rate for each of the approaches. However, Among three approaches, the last approach performs very well and returns the best throughput. In this approach, the system has a detection rate of 100% for known attacks and 96% for unknown attacks for small amount of test data. It means, overall performance for small amount of test data is 98%. For big amount of test data, the system raises some false alarms but this problem is solved with retrain technique. After retraining, false positive rate becomes 0%. However, detection rate for known and unknown attacks are 100% and 90% respectively. Therefore, overall performance for detecting attacks is 95%. The system performance has been compared with that of other related works and

better performance has been reported to some extent.

Chapter 6

Conclusion

6.1 Introduction

The objective of our work was to develop an intrusion detection system capable of detecting anomaly in the computer network. The system was expected to be offline, efficient, automatic, portable, powerful and extensible so that it can be extended to an online system. All of these objectives have been achieved by our system. Different approaches has been adopted to evaluate the system from different point of view. The system performance is quite well and better than some of the similar related systems. Our system achieved a detection rate of 100% for known attacks while the overall detection rate for attacks is 95% with 0% false positives. However, some difficulties has been detected while developing this system. All the difficulties encountered and achievements are given in the following sections. An optimistic future extension is also discussed at the end of this chapter.

6.2 Difficulties Encountered and Limitations

The main difficulty that we have experienced from this work is to train the neural network with huge training data. We have seen from the first approach of training that neural network converge very slowly and it consumes very big amount of time. However, we solve this problem by training the system with small amount of training data and it works better than the system trained with huge training data.

To some extent, our system requires some human involvement. For instance, in case of retraining, we need to merge the data containing anomaly with the training data by hand.

6.3 Achievements

Our system obtains several achievements:

- We propose a set of training approaches for intrusion detection using backpropagation neural networks. Usefulness and drawbacks of these approaches are investigated.
- The architecture of neural network is designed in such a way so that any number of

input and hidden units can be used for the purpose of modification of the neural net for different training approaches.

- The system has been trained with both very big and small amount of training data and the performance analysis shows that big amount of training data is not necessary to train the network for intrusion detection.
- The system is capable of handling huge test data for detection and hence it can be used for a large computer network for intrusion detection.
- The system has a controller to decide the threshold value for detecting attacks.
- The evaluation results show that the system performs better than some of other existing intrusion detection systems.
- Finally, the framework provided by the project work is expected to support significant future extension. The same framework can be used to design an online intrusion detection system.

6.4 Future extension

We have obtained satisfactory achievements from our system as an offline system. This system can be extended to an online system by a little more effort. Currently we can train our system with a very small amount of training data that requires only a few minutes. For testing with huge data, although it takes very little time, it requires some retraining. However, in the current system, the retraining part requires some human involvement. For future extension, first of all, we suggest to make it full automatic so that it does not require any human involvement for the retraining.

Secondly, we suggest to extend this system into an online system. In an online system, the real network is directly connected with the intrusion detection system. The network traffic can be checked by the system periodically for every hour or so. It will only take a couple of minutes for the training and detection of this data of one hour. Now the interesting thing is, if any false positive is detected, the system can be retrained with the data that is responsible for alarming false positive. We have seen from our experience of this work, it takes very small amount of time to be retrained.

However, there is one problem of making such a system. Training data sets may become very large after some time if we always add the data that is responsible for false positives with the training data. To solve this problem, we suggest for multiple weight files. In this technique, a number of weight files will be maintained. If the amount of training data exceeds a predefined number(say, 5000) of training pattern, a new file for training data will be created. The network will be trained with this new data file and new weight files will be created accordingly. Now, we have two sets of weight files. From now on the detection will be performed by these two sets of weight files consecutively. When the new training file is also reached to the ultimate level, another new file will be created and so on, so forth.

For detection, if attacks are detected using the first set of weight files then the particular input patterns that contains those attacks will again be tested with the second set of weight files, and so on until it has been tested with all the weight files. It should be mentioned

here that only those data that carrying the predicted attacks will be used for the next step of testing, not the whole input file. If there is any attack left after checking with all sets of weight files, the system administrator will be notified. However, in this technique, the total number of training data-set files might be increased but the time consumption for detection will be comparatively low. Moreover, as the user behavior can be changed over time, the whole system requires frequent retraining from the very begining. Which, we believe, help keeping the training data sets smaller.

In our work, we only use backpropagation neural network, but there are many other neural networks available. Further research might be carried out to build a system with other types of neural network. Moreover, a combination of more than one types of neural networks can also be used. For reducing the huge training or test input data into smaller amount, data can be grouped together according to some common criteria. The characteristic of those groups instead of actual data can be used for the classification. Kohonen's feature map can be used for such kind of grouping.

Lastly, as an anomaly detector, our system detects only the presence of attacks. It does not tell the name or type of the attack that has been detected. This additional feature can be added to our system by using another classifier e.g. neural network. From our work, we can detect the network packets containing anomaly. So, as a next step, it is possible to feed those packets to another classifier for determining the type of attack.

Appendix A

Instructions for the software

The softwares along with other supportive materials have been provided on a CD.

A.1 Contents of the CD

In addition to the main programs, the CD contains weight files and the test data to evaluate the system. Different types of weight files that generated after training the network for different number of epochs for each of the three approaches have been provided. No training data is provided on the CD. The contents of CD are as follows:

Programs:[1]

preProcess.java , preProcess.class
TrainNet.java, TrainNet.class
ClassNet.java, ClassNet.class
FileInput.class
FileOutput.class

Directories:

Approach_1 contains weight files and raw as well as processed test data for the first approach.

Approach_2 contains weight files and raw as well as processed test data for the second approach.

Approach_3 contains weight files and raw as well as processed test data for the third approach.

A.2 How to run the program

Following are the java commands to run the program:

[1] file I/O libraries are downloaded from http://www.cs.ucl.ac.uk/teaching/DOa1/additnotes/fileoutput.htm

For preprocessing input data

java preProcess *inputFileName outputFileName*

Here, *inputFileName* is the name of tcpdump list file. *outputFileName* is the file name that contains processed tcpdump data. For both training and detection, input files have to be processed using this module.

For Training the Neural Network

java TrainNet

The name of the input files have been given in the program code. The array *trainFiles[]* is holding the name of all input files. It is recommended to change the file name by hand from there, if required. The number of input files that has to be read by the system is maintained by a 'for' loop. For a specific number of input files, it is required that the name of those files are present in the array *trainFiles[]* from the beginning. Moreover, the variable *allFiles* in the 'for' loop should be initialized with the total number of files so that data from all desired files are retrieved. For instance, currently all the name of files from all weeks are present in the array trainFiles[]. If we want to train the net with the files from first 3 weeks, the range for allFiles has to be set from 0 to less than 15. This is because, in one week there are 5 files, so altogether we have 15 files. Currently, This is set in the program by default.

However, The outputs in this case are the weight files. Two weight files *weightIH* and *weightHO* will be generated and stored in the current directory.

For Detection

java ClassNet *fileName*

fileName is the name of file that contains processed input data for detection. This module requires weight values that have to be retrieved from the weight files in the current directory. The weight files are created after training. For testing the system with the provided weight values, it is required that the corresponding weight files are moved to the current directory by hand.

Bibliography

[1] Glossary, Texas State Library and Archives Commission. http://www.tsl.state.tx.us/ld/pubs/compsecurity/glossary.html. Accessed on December 10, 2003.

[2] Bruce Schneier: *Secrets and Lies: Digital Security in a Networked World*, John Wiley & Sons, Inc. 2000.

[3] Edward Amoroso, *Intrusion Detection An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*, First Edition, AT&T Inc. 1999.

[4] Bill Wilson, *The Machine Learning Dictionary for COMP9414*, 1998, 2000. http://www.cse.unsw.edu.au/ billw/mldict.html. Accessed on October 13, 2003.

[5] C. Stergiou, D. Siganos: *Neural Networks.* http://www.doc.ic.ac.uk/ñd/surprise_96/journal/vol4/cs11/report.html. Accessed on July 3, 2003.

[6] Albert Tebo, *Artificial Neural Networks: a developing science*, http://www.spie.org/web/oer/september/neural_net.html, Accessed on December 18, 2003.

[7] *Neural Networks Current Applications*, Edited by P.G.J. Lisboa, pp 35-47, Chapman & Hall, London 1992.

[8] Lou Mendelsohn, *Training Neural Networks*, http://www.profittaker.com/training.asp. Accessed on August 15, 2003.

[9] Sarle, W.S., ed. (1997), *Neural Network FAQ, part 1 of 7: Introduction*, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: ftp://ftp.sas.com/pub/neural/FAQ.html. Accessed on June 25, 2003.

[10] Theuns Verwoerd, Ray Hunt: *Intrusion Detection Techniques and Approaches.* http://www.cosc.canterbury.ac.nz/research/RG/i-net_security/IDSIEEE.pdf, Accessed on November 3, 2003.

[11] Sai Krishna V, *Intrusion Detection Techniques: Pattern Matching and Protocol Analysis*, Security Essentials(GSEC) Version 1.4b April 26, 2003. http://www.giac.org/practical/GSEC/Sai_Krishna_GSEC.pdf. Accessed on December 18, 2003.

[12] Jake Ryan, Meng-Jang Lin, Risto Miikkulainen, *Intrusion Detection with Neural Networks*, Advances in Neural Information Processing Systems 10, Cambridge, MA: MIT Press, 1998. http://citeseer.nj.nec.com/ryan98intrusion.html. Accessed on July 6, 2003.

[13] Peng Ning, Sushil Jajodia, *Intrusion Detection Techniques*, http://discovery.csc.ncsu.edu/p̃ning/Courses/csc774/IDTechniques.pdf. Accessed on December 18, 2003.

[14] David D Elson: *Intrusion Detection, Theory and Practice*, http://www.securityfocus.com/printable/infocus/1203. Accessed on August 2, 2003.

[15] Aurobindo Sundaram: *An Introduction to Intrusion Detection*, http://www.acm.org/crossroads/xrds2-4/intrus.html. Accessed on June 24, 2003.

[16] P. Porras, D. Schnackenberg, S. Staniford-Chen, Davis, M. Stillman, F. Wu: *The Common Intrusion Detection Framework Architecture*, http://www.isi.edu/ brian/cidf/drafts/architecture.txt. Accessed on September 9, 2003.

[17] Charles P. Pfleeger, Shari Lawrence Pfleeger: *Security in Computing*, Third Edition, pp. 468-473, Prentice Hall, 2003.

[18] Srinivas Mukkarnala, Guadalupe Janoske, Andrew Sung, *Intruision Detection Using Neural Networks and Support Vector Machines*, http://www.cs.nmt.edu/ IT/papers/hawaii7.pdf. Accessed on Sepetember 24, 2003.

[19] Alan Bivens, Chandrika Palagiri, Rasheda Smith, Boleslaw Szymanski, Mark Embrechts, *Network-Based Intrusion Detection Using Neural Networks*, http://www.cs.rpi.edu/ szymansk/papers/annie02.pdf. Accessed on June 29, 2003.

[20] Danny Rozenblum: *Understanding Intrusion Detection Systems*, http://www.sans.org/rr/intrusion/understand.php. Accessed on July 14, 2003.

[21] Jeremy Frank: *Artificial Intelligence and Intrusion Detection: Current and Future Directions*
http://seclab.cs.ucdavis.edu/papers/ncsc.94.ps. Accessed on June 10, 2003.

[22] Hervé Debar, Marc Dacier, and Andreas Wespi: *Towards a Taxonomy of Intrusion Detection Systems*, Computer Networks, vol. 31, pp. 805-22, 1999.

[23] Hervé Debar, Marc Dacier, and Andreas Wespi, *A Revised Taxonomy for Intrusion-Detection Systems*, presented at Annales des Télécommunications, vol. 55, 2000, pp. 361-78.

[24] Stefan Axelsson, *Intrusion Detection Systems: A Survey and Taxonomy*, Chalmers University of Technology, Dept. of Computer Engineering, *Göteborg*, Sweden, Technical Report 99-15, http://www.ce.chalmers.se/staff/sax/taxonomy.ps, 2000. Accessed on June 29, 2003.

[25] *Towards a Taxonomy of Intrusion Detection Systems and Attacks*, Research Report RZ 3366, IBM Research, Zurich Research Laboratory, 2001. http://www.newcastle.research.ec.org/maftia/deliverables/D3.pdf. Accessed on October 28, 2003.

[26] Adam Blum, *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*, pp 41-42, John Wiley & Sons, Inc. 1992.

[27] Elaine Rich, Kevin Knight, *Artificial Intelligence* Tata-McGraw Hill Publishing Company Limited, New Delhi, 1996.

[28] Jochen Frohlich, *Neural net overview, the learning process.* http://rfhs8012.fh-regensburg.de/ saj39122/jfroehl/diplom/e-index.html. Accessed on September 18, 2003.

[29] *Backpropagation Algorithm* http://www.macalester.edu/ fox/cs88/backprop.html. Accessed on September 18, 2003.

[30] A.P.Paplinski, *Artificial Neural Networks and their Biological Motivaion*, http://www.csse.monash.edu.au/ app/CSE5301/Lnts/L01.pdf. Accessed on September 1, 2003.

[31] Randall O'Reilly, *How might the brain do backpropagation*, Department of Mathematical and Computer Sciences, Colorado School of Mines. http://www.mines.edu/Academic/macs/colloquia/424.html. Accessed on November 12, 2003.

[32] *Backpropagation Neural Networks.* Gamma Ray Burst Research at Minnesota State University. http://grb.mnsu.edu/grbts/doc/manual/Backpropagation_Neural_Netw.html. Accessed on September 5, 2003.

[33] *Cognitive Psychology, Artificial Neural Networks.* http://sziami.cs.bme.hu/ zady/mi/english/ann3.html. Accessed on August 03, 2003.

[34] Robert J. shimonski, *What You Need to Know About Intrusion Detection Systems.* http://www.windowsecurity.com/articles/intrusion_detection/ Accessed on August 21, 2003.

[35] DARPA Intrusion Detection Evaluation, Lincoln Laboratory, Massachusetts Institute of Technology. http://www.ll.mit.edu/IST/ideval/index.html. Accessed on July 15, 2003.

[36] DARPA Intrusion Detection Evaluation Data Sets for 1998, Lincoln Laboratory, Massachusetts Institute of Technology. http://www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.html. Accessed on July 17, 2003.

[37] Morten Bek, Troels Grosbøll-Poulsen, M. U. Kristoffersen, *Evolutionary Trained Kohonen Networks as Classifiers for Human Utterances*, Department of Computer Science, University of Aarhus, Denamrk, May 2002. http://www.daimi.au.dk/ bek/thesis_html/thesis.html. Accessed on November 15, 2003.

[38] Robert Birkely, *A Neural Network Based Intelligent Intrusion Detection System*, Masters thesis, School of Information Technology, Faculty of Engineering and Information Technology, Griffith University-Gold Cost Campus, June 2003. http://www.rbirkely.com/CV/Intelligent_Intrusion_Detection_System.pdf. Accessed on December 12, 2003.