Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Wireless Local Area Network Intrusion Detection System Using Deep Belief Networks

By: Temesgen Mihiretu Abebe

April 23, 2018

A thesis submitted to the School of Electrical and Computer Engineering in partial fulfillment of the requirements for the Degree of Master of Science in Computer Engineering

By: Temesgen Mihiretu Abebe

Advisor: Menore Tekeba

April 23, 2018

Addis Ababa University
Addis Ababa Institute of Technology
School of Electrical and Computer Engineering

Wireless Local Area Network Intrusion Detection System Using Deep Belief Networks

---

By

Temesgen Mihiretu Abebe

Advisor

Menore Tekeba

April 23, 2018

Declaration

I,the undersigned, certify that research work titled Wireless Local Area Network Intrusion Detection System using Deep Belief Networks is my own work. The work has not been presented elsewhere for assessment. Where material has been used from other sources, it has been properly acknowledged.

Temesgen Mihiretu Abebe signature _____

Date of submission: April 23, 2018

Place: Addis Ababa

This thesis has been submitted for examination with my approval as a university advisor.

Advisor: Menore Tekeba signature _____

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Wireless Local Area Network Intrusion Detection System Using Deep Belief Networks

By

Temesgen Mihiretu Abebe

Approval By Board of Examiners

Dr. Yalemzewd Negash	_____
Dean, School of Electrical and Computer Engineering	Signature
Menore Tekeba	_____
Advisor	Signature
_____	_____
Internal Examiner	Signature
_____	_____
External Examiner	Signature

Abstract

In computer security, Intrusion detection Systems (IDS) are mechanism of detecting an intruder in the system and notifying malicious activities to system administrator. Most of IDS researches are on wired Local Area network (LAN) using KDD dataset. But the wireless IDS needs its own research using dataset from wireless LAN. Since most of the security vulnerability features of wireless LAN is due of its nature and they are different from wired LAN, wireless IDS needs to be studied independently from that of wired LAN. The IDS researches on wireless LAN started recently. Until now there are some research works like publishing Aegean Wi-Fi Intrusion Dataset (AWID) dataset publicly for the research community and evaluating the dataset using different machine learning algorithms. But when we see the results from the previous research works, especially in the case on Flooding and Impersonation attacks, it is clear that wireless IDS is not well researched and it needs further study for performance improvements.

The AWID dataset contains different data types which are numeric, string, and hexadecimals. So before training the system and evaluation of its performance, the dataset is preprocessed and finally 102 attributes are used for system training and evaluation. Also two stage feature selection is implemented to reduce the training cost and improve the system performance by selecting the minimum number of most discriminant features. The first stage is removing duplicated attributes, which reduce the number of attributes in the dataset to 68. The second stage is done by applying Information Gain Ratio (IGR). Using three thresholds three dataset are prepared namely 41 attribute dataset, 34 attribute dataset, and 25 attribute dataset to experiment the relation between number of attributes in the dataset and the resulting system performance. The main classification system is implemented using Deep Belief Networks (DBN). Two stage training strategy is

used to train DBN for classification. The first stage is unsupervised pre-training using Restricted Boltzmann Machine (RBM) and the second stage is supervised fine tuning of the pretrained DBN parameters using Back Propagation Neural Network (BPNN) algorithm.

Finally after designing and implementing the system, a number of experiments have been done to evaluate the system performance using different performance metrics. The system was able to achieve 98.55% classification accuracy with 102 attributes and it was able to improve this result to 98.97% with selected 34 attribute dataset evaluation. But the classification accuracy decrease to 98.74% while the numbers of attributes decrease to 25. This shows that there is a limit in reducing the number of attributes and from the experiments it is found that the minimum number of the most discriminative attribute list that was able to reach the maximum performance in the proposed system is 34 attributes. The system has been tested also using 10-fold cross validation and its classification accuracy was improved to 99.96%.

Keywords- *AWID dataset, wireless network, intrusion detection system, Deep Belief Networks, attribute selection.*

Acknowledgments

First of all I would like to thank the Almighty God for giving me all the strength throughout my life including this thesis challenging journey.

Second, my deepest gratitude is to my advisor, Menore Tekeba, for his encouragement, guidance, support, and enthusiasm with his knowledge. His guidance helped me in all the time of research and writing of this thesis.

Third, I would like to thank Quamar Niyaz, who is PhD candidate at University of Toledo, for his fruitful information about AWID dataset.

Last, but not least, I would like to offer my regards and blessings to my entire family members including my wife Melkam Animaw, my son Ahadu Temesgen, and my mother Wubalech Abebe, friends and those who supported me in any respect during the completion of the thesis.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
ADNIDS	**A**nomaly **D**etection based **N**etwork **I**ntrusion **D**etection **S**ystem
AI	**A**rtificial **I**ntelligence
ARP	**A**ddress **R**esolution **P**rotocol
AP	**A**ccess **P**oint
AWID	**A**egean **Wi-Fi** **I**ntrusion **D**ataset
BPNN	**B**ack **P**ropagation **N**eural **N**etwork
CSMA/CA	**C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance
CD	**C**onstructive **D**ivergence
DBN	**D**eep **B**elief **N**etwork
ESSID	**E**xtended **S**ervice **S**et **I**dentifier
DoS	**D**enial of **S**ervice
HIDS	**H**ost based **I**ntrusion **D**etection **S**ystem
IDS	**I**ntrusion **D**etection **S**ystem
IGR	**I**nformation **G**ain **R**atio
LAN	**L**ocal **A**rea **N**etwork
MAC	**M**edia **A**ccess **C**ontrol
NaN	**N**ot a **N**umber

NIDS	**N**etwork based **I**ntrusion **D**etection **S**ystem
PSK	**P**re **S**hared **K**ey
RBM	**R**estricted **B**oltzmann **M**achine
SAE	**S**tacked **A**uto **E**ncoder
SNIDS	**S**ignature based **N**etwork **I**ntrusion **D**etection **S**ystem
SSID	**S**ervice **S**et **ID**entifier
SVM	**S**upport **V**ector **M**achine
WEP	**W**ired **E**quivalent **P**rivacy
Wi-Fi	**W**ireless **F**idelity
WPA	**Wi-Fi P**rotocol **A**ccess

Chapter 1

Introduction

1.1 Background

In the history of computing and networking, there are different security threats and they are increasing from time to time in their types and forms. Most of the security measure to protect computers and networks from the intended threats can be categorized as protection and detection mechanisms. Protection mechanisms are the way of protecting the system from the intended threat by taking different actions. But they are not enough to secure the system all the time and in all scenarios. That is why detection and monitoring mechanisms are as necessary as protection mechanisms to alert the system about security threats by tracking the system activities and network traffic. An IDS is a security system that monitors computer systems and network traffic and analyzes that traffic for possible hostile attacks originating from outside the organization and also for system misuse or attacks originating from inside the organization. There are broadly two types of IDSs. These are host based IDS (HIDS) and network based IDS(NIDS). HIDS monitor the host system activities and NIDS monitor the network traffics [1].

Based on the methods of intrusion detection mechanism they used to detect malicious traffic, NIDS can be classified as signature (misuse) based NIDS (SNIDS) and anomaly detection based NIDS (ADNIDS) [2]. In the case SNIDS rules for the specific type of attack are pre-installed in the intended network environment, then the IDS based on the

specified rule it detects the signature of the traffic for the specified attack. In the case of ADNIDS whenever deviation from the normal traffic is observed it will be classified as intruders.

In the computer network, Wireless Fidelity (Wi-Fi) is a wireless connection type other than wired LAN. Wi-Fi is recently growing technology for local area connection because of it is easy and less expensive to implement than wired LAN. But it is vulnerable to various attacks and intruders because of its security vulnerable features [3–6]. So studying IDS for wireless network independently from wired LAN gives better advantage for wireless LAN security.

Anomaly detection could be done using different mechanism such as using different Artificial Intelligence (AI) algorithms, using statistics methods, data-miming based, etc. and there are different researches and implementation using such methodologies. Among them AI based IDS research is the main focus of this research. AI is one of modern computing mechanism by dreaming to achieve human level intelligence in computing world. Since the beginning until now it was not left as a dream, but it was able to achieve big success in different fields of study and also it is evolving in its domain for better outcome based on new thought in the area and its experience in the field of practice. Deep learning is one of AI under machine learning field. It is one of the most recent study of AI for better performance to reach the promise of the AI. Deep learning is the way of multi-layer network machine learning architecture [7]. It include different algorithms, among them DBN is recently proved to be very effective for a variety of machine learning problems [1, 8].

Machine learning algorithms try to model the intended system based on sample or observation of the system activities. Datasets are the way of sampling the intended system working environment and they are used by machine learning algorithms to model the system. The dataset may be prepared by the researcher himself or the researcher may use scientifically collected and published dataset for their research. There are different dataset out there for network IDS research by different researchers and organizations.

Among them NSL-KDD [9] and AWID [3, 10] are the most usable dataset for wired LAN and wireless LAN respectively. NSL-KKDD dataset is collected from LAN by DARPA and AWID is collected form Wired Equivalent Privacy (WEP) protected 802.11 wireless networks by University of the Aegean. Both are labeled dataset for their respective class of attacks and normal traffic. Since this thesis is on wireless IDS , the thesis is focused on implementing DBN for wireless IDS and AWID dataset will be used to create the model of the proposed system through training and evaluate the model about its performance.

1.2 Problem Statement

There are different tool and researches regarding IDS based on different approach like rule based, data mining techniques, machine learning algorithms, Support Vector Machine (SVM), etc. But most of the researches are done for wired network. Since the vulnerability and attack type of wireless network is different from that of wired network as discussed in the research work [4–6, 11–14], there should be an independent equivalent research practice for wireless network intrusion detection systems. But when we see the research practice on wireless network IDS, the research area is still young and under development. Among the main reason for this under-developed research, one is wireless technology new; there has not any standard dataset before AWID for wireless network. So most of the studies are based on different version of KDD dataset which is collected from the wired LAN or the data collected by the researchers.

On wireless IDS after AWID dataset there are some work using this dataset using different machine learning techniques, also some of them include the advantage of feature reduction in their work [3, 15–17]. The main problem is, even after all those research works, the performance of the wireless IDS is poor and it needs some improvement espetially in the case of Flooding and Impersonation attacks. So since DBN is proven to be the best deep learning classifier in some recent works like [1, 8] , evaluating AWID dataset using DBN and also studying the effect of feature selection on the performance of the implemented system is very important and challenging task as the wireless network IDS domain. So

the research questions of this thesis are listed as follows.

- Can DBN algorithm improve the performance of Wireless network IDS system?
- Does feature selection on the dataset attributes improve the performance of wireless IDS?

1.3 Objective

General Objective

The general objective of the thesis is to design and implement wireless network IDS using DBN machine learning algorithm and assess the performance of the developed model.

Specific Objective

Under the above general objective the thesis has the following specific objectives:

- To implement proper preprocessing methods for dataset preprocessing.
- To train DBN using AWID dataset for wireless IDS classification and assess its performance using different performance evaluation metrics.
- To test the effect of feature selection on detection performance and computational cost of the proposed wireless IDS.
- To test the effect of 10-fold cross validation on the performance of the proposed wireless IDS.
- To compare the proposed system performance with similar previous works.

1.4 Methodology

The methodology that was followed for this thesis study is organized in to collecting the dataset from its source specifically its website [10], literature review, dataset prepro-

cessing, feature selection, 10-fold cross validation data preparation, system design and implementation, system training and performance testing, and finally documenting the result. The most basic methodologies are discussed as follows.

1.4.1 Literature Review

In this step of the thesis different literature was researched and reviewed which are more related for the system. The review area is research on wired LAN intrusion detection systems, AWID dataset, and wireless IDS using AWID based on machine learning techniques. The researches were selected based on their publisher rank, citation index, time of publication, and their achievement. So almost all the documents are supposed to be up to date, high ranked and have good achievement as much as possible.

1.4.2 Preprocessing and feature selection

AWID dataset was used for training and testing purpose for the implemented system. The dataset was selected from the list of AWID datasets and preprocessed using different preprocessing techniques. Then feature selection is implemented to select the most discriminant feature set of the dataset for the proposed system.

1.4.3 System design and implementation

The design include figuring out the architecture of the system, determining the training approach of each component of the system, determining system parameters and their value, and specifying the performance metrics of the system. The implementation part is just how to implement the system from different perspective like development language, development resource, etc.

1.4.4 Result Presentation and Discussion

In this stage result from the implemented system training and testing phase is collected and presented in more descriptive way and there will be detailed discussion on the pre-

sented data. The presentation technique includes tables, graphs, and other appropriate mechanisms. The discussion contains more scientific explanation and comparison with the previous works.

1.5 Scope

Wireless IDS is an IDS for wireless network. It could be implemented using different computation methodologies like role based, data-mining techniques, machine learning, etc. The dataset for the study could come from different source like prepared by researchers or from publicly available scientifically published datasets.

Since it is difficult to study every aspect of wireless network IDS, this thesis has the following scopes and limitations.

- It uses high label reduced (AWID-CLS-R) dataset among list of AWID datasets for training and evaluation purpose of the system.
- It is implemented using DBN with unsupervised greedy pre training and BPNN for fine tuning algorithms.
- It use one chosen feature selection algorithm only to see the effect of feature selection on the performance of the system.
- It implements 10-fold cross validation in addition to the usual hold out cross validation for better system training and performance testing.
- The system is able to work only offline on pre-processed dataset. It needs some work to integrate the system in real live online environment.
- It analyzes and discusses the experimental results, it compares the result with previous work and it gives scientific conclusion for the resulting system.

1.6 Contribution of the research

Research on IDS is ongoing process, because networking technology are fast growing and changing technology and the attack and threats so does. The case is worth on wireless networking technology since it is new, growing and more vulnerable for attacks than the wireless networking due to its nature. Until now the researches on wireless IDS was able to prepare publicly available dataset [3] and it was able to evaluate the dataset using different machine learning algorithms [3, 15, 16]. But there is no research that evaluates the dataset using deep learning algorithms specifically using DBN algorithm. So the main contribution of this thesis is the evaluation of AWID dataset using DBN algorithm for performance improvement.

1.7 Thesis Outline

The organization of this thesis documentation is described as follows from the next chapter two up to the end chapter. Chapter 2 discuss in detail about background knowledges and different literatures of intrusion detection, like Wi-Fi network, AWID dataset, DBN and its training methodologies. Chapter 3 discuss on the design and implementation of the system starting from normalization algorithm up to classification system. Then chapter 4 discusses the experiments and the results based in the design and implementation as discussed in chapter 3. Chapter 5 is the final chapter of this documentation and it will discuss about the conclusion and recommendation of the thesis in general.

Also this documentation contains abstract of the thesis, declaration, acknowledgment, and introduction in the beginning of the document. At the end of the document next to chapter 5, it contain reference of different literature that are used in the thesis and appendix for system implementation code.

Chapter 2

Theoretical Background And Literature Reviews

In this chapter the theoretical background of the system is described and the literature reviews are made on different IDS researches that are more related to the proposed system. The theoretical background will focus on wireless LAN, wireless IDS, and DBN machine learning algorithm.

2.1 802.11 Wireless Local Area Network

Wireless network known as Wi-Fi is an IEEE 802.11 specification that governs wireless network communication methods. It use radio frequency for half duplex communication between the transmitting and receiving devices. wireless LAN is flexible, easy to deploy with little cost and easy to mobility than wired LAN [4, 18].

Wireless network can be categorized as infrastructure based and Ad-Hoc in their architecture [3]. The infrastructure based wireless network use Access Point (AP), which connects to the backbone network, for data transmission by encoding the data as radio frequency and the receiver device use their wireless antenna device to receive the transmitted signal and decode it to the transmitted data as shown in the figure 2.1 which is taken from [19]. In the Ad-Hoc case the wireless clients directly connect together without any wireless

router or AP. Each node in Ad-Hoc network used as receiver and transmitting router in the network. Since AWID dataset is collected from infrastructure based wireless LAN as discussed in the research work [3], in this thesis the main focus is on infrastructure based wireless LAN.

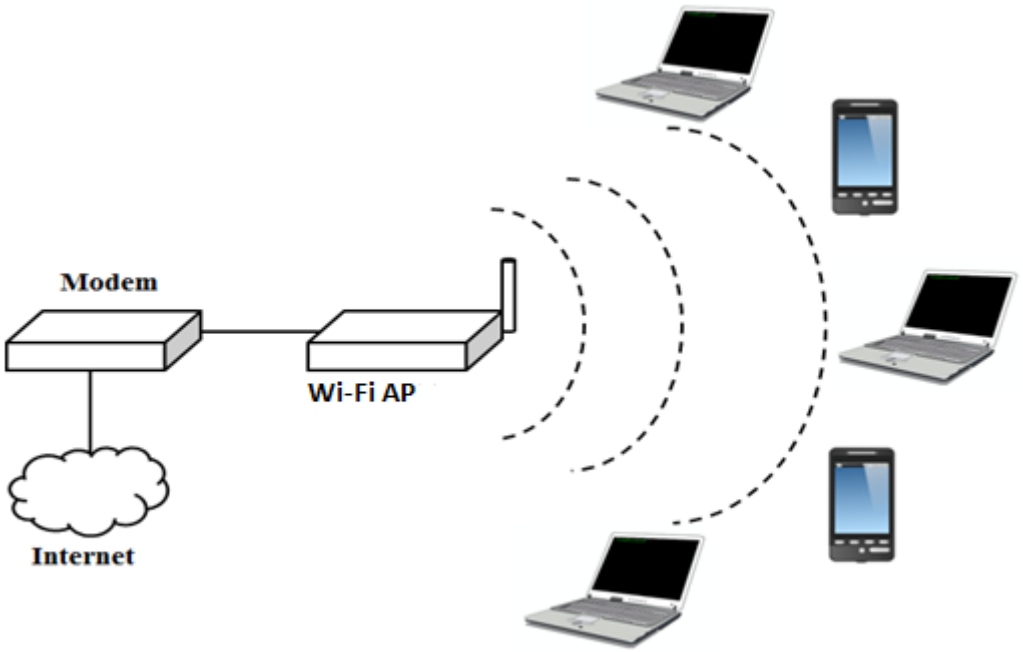


Figure 2.1: Infrastructure based wireless communication

In 802.11 there are three types of frames, which are management frame, control frame, and data frame [3]. Each of them has different length and field based on their respective purpose.

The management frame in 802.11 used to establish and maintain communication between AP and clients. It perform supervisory functions that are join and leave wireless networks and move associations from AP to AP [3, 5]. It contains different subtypes based on its purpose. The most common subtypes are authentication, de-authentication, association request and response, re-association request and response, beacon, Disassociation, Beacon, Probe Request, and Probe Response [3].

The second type of frame is control frame that work on the delivery of data frame between AP and clients. It performs area clearing operations, channel acquisition, carrier-sensing

maintenance functions, and positive acknowledgment of received data [3]. It only contains header information. It must be heard by all the stations; therefore they must be transmitted at one of the basic rates. It has the following subtypes, which are Request to Send (RTS), Clear to Send (CTS), Acknowledgement, Power Save (PS) Poll.

The third type of frame is data frame, which is the main frame to transmit the actual data from higher layers between the communication Medias [3]. Based on different function, such as contention based or free service, quality of service, and information overloading, it has different type.

Wireless network has its own group of specification under IEEE 802.11. 802.11b was the first and the most widely deployed wireless standard [6]. It use 2.4 GHz unlicensed radio band which could deliver the maximum of 11Mbs data rate. Since it is half duplex communication it use carrier sense multiple access with collision avoidance (CSMA/CA) mechanism for collision detection. 802.11g is the next standard that can delivers 54Mbps data rate with 2.4GHz frequency [18]. Also it is backward compatible with 802.11a and they are able to operation together in the same network. But the network should run 802.11b CSMA/CA collision detection method for both standards which results performance degradation. 802.11a is the other standard delivers a maximum of 54Mbps data rate and operates on 5GHz band of frequency . It is not compatible with 802.11b standard because they operate on a different frequency band.

2.2 Wireless Security Threats and Vulnerability

From the first version of 802.11 amendments until now security is considered as the main component of the design. In general there are three well known security mechanism in wireless LAN.

The first security mechanism was WEP for the first version of 802.11 of protocol [3, 20]. The main purpose was to provide confidentiality of the wireless network comparable to that of provided by wired network. It use RC4 encryption algorithm. But through time

it was proved that is susceptible to different attacks and it is replaced by the next version of security mechanism. But until now lots of wireless connection use it as their security mechanism.

The next version of security mechanism was Wi-Fi Protocol Access (WPA). WPA was intended to solve problem of WEP without the need to change their device [11]. It uses more complex data encryption algorithms Temporal Key Integrity Protocol (TKIP) or Advanced Encryption Standard. Then the current security mechanism is WPA2 to increase the security of wireless network, which is part of IEEE 802.11i version of the IEEE 802.11 standard [3].

Wireless LAN is vulnerable to different security threats due to its nature. There are number of research works which analyze the security issue of the standard from different perspective.

The research work in [12],describes how wireless LAN encryption standards are vulnerable to different attacks such as Chop-chop attack, Brute force attack, Beck-Tews, Halvorsen-Haugen and 196 attacks. The chop-chop attack exploits WEP encryption by determining the Pre-Shared Key (PSK) through trial and error, other than mathematically or cryptographically methodologies. The brute force attack uses a library of possible PSKs to find a match for a captured four way handshake.

Since WPA reuse WEP, both are vulnerable to offline dictionary attack and Denial-of-Service (DoS) attack which is done over layer 2 by using de-authentication and disassociation messages to the client or AP resulting in the legitimate user being denied access to the service [13, 14]. Also they are vulnerable to dictionary and brute-force attack due to pre-shared key authentication [4].

In all security protocols all encryptions are applied only for data frame not on management and control frames. So all 802.11 management and control frames are vulnerable

to different attacks like probe, associate, authenticates, disassociate, and de-authenticate attacks [3, 4].

In research work [5], it is clearly stated that wireless LANs are vulnerable to radio frequency interference problems. Since it is almost impossible to effectively contain the radio frequency signal it makes wireless LAN vulnerable to different attacks such as it makes easier for attacker to sniff sensitive datas [4]. Also even though it is possible to configure Media Access Control (MAC) address filtering at AP it is possible for attacker to sniff large amount of traffic including MAC address and change their MAC address to act as legitimate client [4]. The other thing is than sometime Service Set Identifier (SSID) are used as their default value which result easy penetration of the network [6].

All the above literatures shows that how wireless LAN is susceptible to attack which targets to breaks confidentiality and integrity of datas and network availability. The vulnerability features which are discussed in the researches are based the nature of the network being wireless and its communication mechanism. So it is possible to say that most of the security vulnerability and threats come from its nature of being wireless, communication protocol, etc. which makes wireless network vulnerability different from that of wired LAN.

2.3 Literature Reviews on Related Works

Intrusion detection system in computer network has been the active research field for a long period of time. There are so many researches by different researchers which have been done for a long period of time until now. Intrusion detection system researches can be classified as rule based IDS which use pre-installed specific rule to detect intruders like Snort and intelligent IDS in which trained IDS used to detect intruders like machine learning algorithms. Most of intelligent IDS researches are based on large amount data for performance benchmarking. Researcher may prepare the dataset by themselves from the real time trace or using simulation or may use standard common datasets. Intrusion detection systems that are based on standard common datasets are easier to compare its

performance achievements and repeat the experiments. The common standard datasets for IDS are NSL-KDD dataset for wired LAN and AWID dataset for wireless LAN which are the two freely available datasets for IDS research practice. Since it is impossible to review all the research works, recent IDS based on machine learning algorithms with standard datasets as performance benchmark are reviewed to be a millstone for the proposed research in different perspectives. The literatures include IDS on wired LAN and IDS on wireless LAN.

2.3.1 Intrusion Detection System on Wired Network

For wired network IDS, the common dataset used is NSL-KKD dataset. NSL-KDD dataset is continuation and improved set of its predecessor DABRA 98 and KDD Cup 99 datasets regarding system accuracy and reduced false positive [1, 9]. The dataset is collected by MIT Lincoln Labs from LAN which simulates typical U.S. Air Force LAN [21]. The recorded traffic is categorized to four main types of attacks and normal classes. The attack types are DoS, Probe, Remote to Local (R2L), and User to Root (U2R). It has 41 features set [21].

The research in [22] is based on SVM for classification and NSL-KDD dataset as performance benchmark. Since the performance of SVM is affected by the number of data dimensionality and it treats every data equally they use IGR for feature ranking and K-mean classifiers predictive accuracy feature selection during SVM training to maximize the detection rate [22]. Finally they found, from their experimental analysis, feature reduction maximizes the performance of the classifier and reduce computational cost [22].

The other research which works on improving the number of support vectors in SVM and reduction of training time in [23] uses Modified SVM for classification with NSL-KDD dataset. The Modified SVM is the combination of SVM and K-Nearest-Neighbor [23]. Using Modified SVM, they improved the number of support vectors and amount of required training time while keeping comparable detection accuracy [23].

The research work in [24] is based on hybrid classifier which combines DBN for feature reduction and SVM for classification of the traffic to normal and attacks. For evaluation purpose they use NSL-KDD dataset. They compare the performance of DBN-SVM hybrid with SVM performance and their experimental result shows that the proposed hybrid classifier system outperforms SVM classifiers [24]. But the execution time of their hybrid system is greater than both classifiers. Also, DBN as a feature reduction method is compared with conventional Principal Component Analysis (PCA), Gain Ratio and chi square to reduce the 41 features of the NSL- KDD dataset to 13 features [24]. Then they have evaluated the performance of the feature reduction based on the SVM classifier accuracy and their experimental result shows that DBN perform better than the other mentioned feature reduction techniques [24]

The deep learning approach based work in [2] is the implementation of IDS using self-taught learning (STL) approach. They use sparse Auto-encoder; because of its easy to implement even though it is better to use other algorithms like stacked auto-encoder; for good feature representation from large collection of unlabeled which is termed as unsupervised feature learning (UFL) and they use soft-max regression (SMR) for the classification task. They choose sparse Auto-encoder and for their work they have used NSL-KDD dataset. Finally they also compare their work with some previous works and they found a better result than the previous similar works [2].

The more recent IDS is the research work in [1], which uses DBN for normal and attack class classification. The DBN they used for classification is stack of RBM for the intrusion detection system. The work is evaluated using reduced NSL-KDD with Numerical encoding. The feature set they used for evaluation is 39 out of original 41 to remove Not a number (NaN) due to zero valued features [1] after preprocessing of the dataset. They compare their work with previous work DBN-SVM hybrid system and the experimental result shows that their proposed DBN based system perform better than its former work [1]. But their feature selection technique is not clearly stated.

2.3.2 Intrusion Detection System on Wireless Network

For wireless LAN IDS, the research work [3] is publication of public dataset by collecting from 802.11 wireless fidelity (Wi-Fi) networks which contain normal and attack traffics. Even though the dataset is collected from WEP protected network, the researcher states that the same dataset is possible to be used for WPA/WPA2 protected network intrusion detection research, since the existing threats of the latter are practically a subset of the first [3]. Their work can be seen in two perspectives. The first one is they contribute a publicly available labeled dataset for IDS on Wi-Fi network research community [3]. The second one is they gather, categorizes, and evaluate the most known attack from 802.11 WEP protected Wi-Fi network standard [3].

They evaluate the full feature and reduced 20 feature datasets with the following machine learning classification algorithms which are Adaboost, J48, ZeroR, OneR, Random Tree, Random Forest, Nave Bayes, and Hyperpipes [3]. Then for both feature sets J48 outperform the other classification algorithms and the reduced 20 feature dataset perform better than the 154 featured dataset [3]. They also showed that feature selection and avoiding feature redundancy have great significance on the detection performance of intrusion detection systems [3]. But the feature selection algorithm they have used is not clearly specified.

The research in [15] works on the evaluation of AWID dataset using machine learning algorithms with Information Gain (GI) and Chi Squared statistics (CH) filter based feature selection techniques. Their main aim was to show the effect of feature selection on intrusion detection dataset on the performance of IDS attack detection [15]. Finally they show that feature reduction can improve the detection accuracy and classification speed, but when they further reduce the number of features, it can decrease the detection accuracy [15]. The machine learning technique they used for evaluation was OneR, J48, Random Forest, Random Tree, and Ada Boost; and from their experimental result Random Tree perform better for high level class distribution and Random forest perform better for finer grain class distribution [15].

The research in [16] implements majority voting technique for wireless IDS with data mining technique for feature selection which reduce the feature set to 20. The majority voting is combination of Extra Trees of 20 trees, Random Forests of 20 trees, and the Bagging classifier of 10 Decision Trees as base estimator [16]. They implement Decision Trees, Extra Trees, and Random Forests for all feature set and reduced feature set and they found that none of them performed better then J48 which is best performer in [3], also decision tree was not stable at all [16]. Then they implement bagging classifier for all feature set and reduced feature set and it gives slightly better performance with better timing [16]. Finally they implement their proposed system for all feature set and reduced feature set and they found that it have better than the above tested techniques and equal for all feature and reduced feature dataset performance but the reduced dataset reduce the execution time [16].

The work in [17] focused on improving the detection level of Impersonation attack which was insignificance in [1]. They use Artificial Neural Network (ANN) for feature selection and Stacked Auto Encoder (SAE) for classification and they evaluate their proposed system using AWID-CLS-R for both training and testing [17]. Also they prepare the balanced version of the dataset in their preprocessing for training purpose since the dataset contains a huge number of normal instances compared to attack instances, especially Impersonation Attack [17]. Finally from their experimental result they showed that the performance improvements of Impersonation attack as they propose first.

As a summary; all the above research work shows that what was accomplish in IDS research and it needs additional works to lead the research to more improvement of its performance. In section 2.3.1, wired LAN IDS which are based on family of KDD dataset was reviewed and the research [2] is the most recent and the most successful from its kind. But is use KDD dataset as performance benchmark which is collected from wired LAN [9, 25] which shows that wireless LAN IDS should be study independent from that of wired LAN research with its own dataset.In section 2.3.2reviews wireless LAN IDS starting from the AWID dataset collection and its evaluation by different researchers. In both sections the research which applies feature selection methods gets better per-

formance, which shows the advantage of feature selection in the performance of IDS. So after reviewing all the above researches it is possible to conclude that none of the above reviewed research implements wireless LAN IDS using DBN as classifier and AWID as performance evaluation.

2.4 Dataset

For machine learning based system dataset is the main concern; because the algorithms are trying to model the system through the dataset. So dataset is expected to be representative of the real operational environment. As mentioned in section 1.1, for IDS study there are two publicly available datasets which are KDD dataset and AWID dataset. Since the main concern of this thesis is wireless IDS then this section will discuss on AWID dataset.

In general AWID dataset is collection of different wireless datasets independently using different sessions [3]. Based on the attack class labeling there are two categories which are high label (AWID-CLS) and finer grained (AWID-ATK). The high level contains four classes namely Normal, Injection, Impersonation, and Flooding in both training testing dataset and the finer grain version contains 16 classes.

Then based on their size both contain full dataset which is tagged as F namely AWID-ATK-F and AWID-CLS-F and reduced dataset which is tagged as R namely AWID-ATK-R and AWID-CLS-R. Each of them has separate training and testing dataset [3]. Totally the dataset contain four datasets as shown in the figure 2.2. AWID dataset contains 155 attributes [1] [2] including the attack class labeling [3, 26]. The features are composed of mainly MAC layer information like source address, destination address, initialization vector (IV), SSID, etc. the rest of attributes contain Radio tap information, general frame information and frame number [3].

[1] Unless exclusively stated that it is including the class labeling when number of attributes are expressed it means number of attributes in the dataset without the class labeling column.

[2] The word attribute and feature are interchangeably used in this document while referring the dataset columns

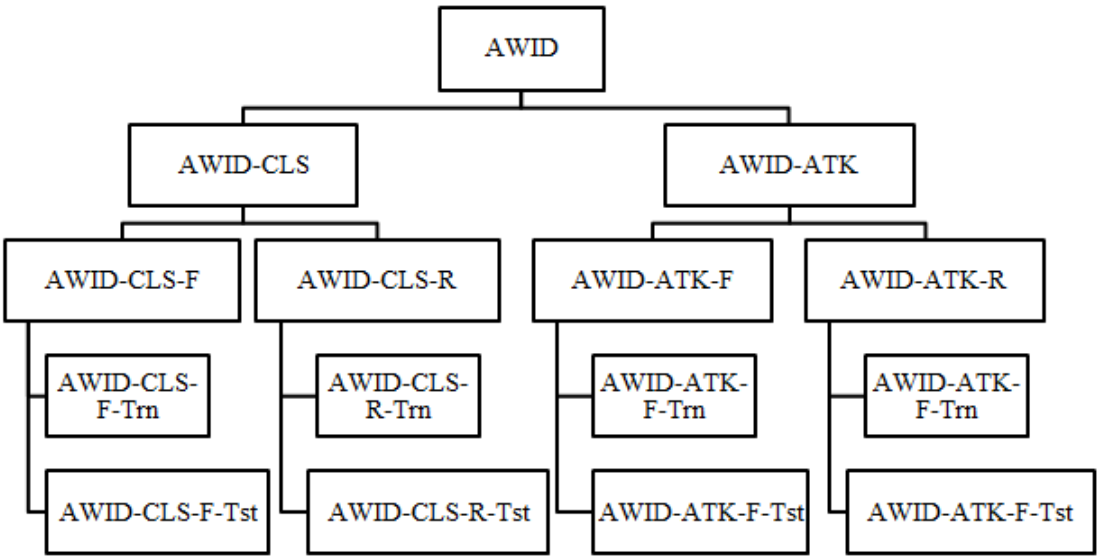


Figure 2.2: AWID dataset structure

2.4.1 Wireless Local Area Network Attacks and Their Classification Methods

While the research work in [3] is preparing the dataset, they test and study around 24 attacks based on their conceptual similarities. First they study those attacks by grouping them using their attack purposes in to key retrieving attack, Key stream Retrieving Attacks, availability attacks and Man-in-the-Middle Attacks. Key retrieving attacks tries to crack the Secret Key offline by monitoring specific packets first. It is passive type of attack; so it is untraceable unless the attacker tries to execute the active version of the attack by injecting a large number of packets in the network. Key stream Retrieving Attacks focuses on retrieving key stream that can be used to create other attacks through injection. Availability attacks are commonly known as DoS attack since the attacks try to deny the system service by consuming or blocking the resource in the network. IN the case of Man-in-the-Middle, attackers try to create fake network and then they advertise luring or existing ESSID to fool the client to connect with them instead on the legitimate network to monitor their network traffic.

But since IDS system try to infer common patterns among the attack of the same class, the researchers in the research work [3] categorize the attacks according to the attack

methodologies of execution. So the new groups of attack are Injection, Flooding, Impersonation, and passive attack. All the attacks they studied are grouped under this list of attacks especially in the high label (CLS) version of the dataset as shown in the table 2.1. In Injection attack, packets are injected in the normal traffic and it causes overflows

Table 2.1: LAN attacks and their classification

Attack Class	**Attack**	**Purpose**	**Target**
Injection	ARP Injection, Chop-Chop, Fragmentation	Key Cracking	Network
Flooding	Deauthentication, Disassociation, Disassociation , Deauthentication broadcast, Disassociation broadcast, Block Acknowledge, Authentication Request, Fake Power Saving, CTS, RTS, Beacon , Probe Request, Probe Response	DoS	client
Impersonation	Caffe Latte, Hirte, Honeypot, Evil Twin	Keystream, Man-in-the-Middle	Network, Client
Passive	FMS, Korek, PTW, Dictionary	Key Cracking	Network

in the traffic size. They are key cracking attacks and they target the network [3]. Flooding attacks are DoS attacks that create a sudden increase in the management frames in certain unit of time by targeting the client side of the system. Impersonation attacks introduce an additional AP in the neighborhood by broadcasting Beacon frames that advertise a preexisting valid network and it double the number of Beacon frames of the victim network. Then with the help of DE-authentication frames as an initial step the attacker force the clients to connect to their false AP. Passive attacks are a kind of key cracking attack that target the network. From the above four type of attacks the first three are active attacks which can be track for their hostile activities in the system and the last one as its name implies, it is a passive type of attack.

2.5 Deep Belief Networks and Training Algorithms

2.5.1 Deep Belief Networks

Deep Belief Networks is probabilistic generative models which are composed of multiple layers of latent stochastic variables [8, 27]. It is a graphical model as shown in figure 2.3, which learns to extract a deep hierarchical representation of training data. The top two layers form an undirected bipartite boltzmann machines while the lower layers form a directed sigmoid belief network. The training process consists of two steps. The first one is unsupervised pre training and the second one is supervised fine-tuning. In [28] it is clearly stated that a DBN is a stack of RBM and it can train each of RBM in DBN in unsupervised greedy manner for DBN unsupervised pre-training. Then the next step is supervised fine-tuning of the same structure using different algorithm like BPNN machine learning algorithm. The joint distribution of DBN between the input x and

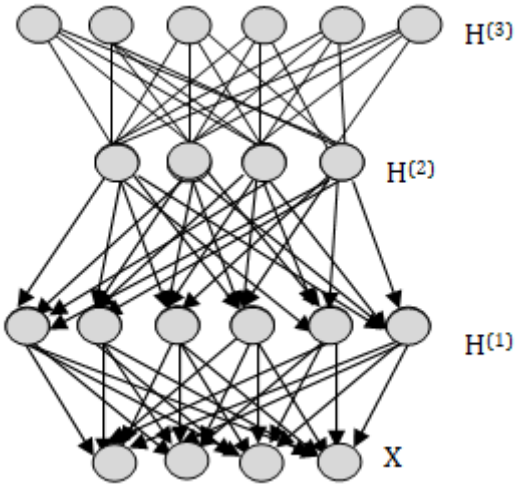


Figure 2.3: Deep Belief Networks

hidden variable g^i at layer l of L layers is represented as follows in equation (2.1).

$$P(x, g^1, g^2, ...g^l) = P(x|g^1)P(g^1|g^2)...P(g^{l-2}|g^{l-1})P(g^{l-1}|g^l) \tag{2.1}$$

Where $P(g^i|g^{i+1})$ is the conditional probability for the given two hidden layers and can be represented as follow in equation (2.2) and (2.3).

$$P(g^i|g^{i+1}) = \prod_{j=1}^{n^i} p(g^i|g^{i+1}) \tag{2.2}$$

$$P(g_j^i = 1|g^{i+1}) = \sigma(b_j^i) + \sum_k^{n^{i+1}} W_{kj}^i g_k^{i+1} \tag{2.3}$$

where:

- b_j^i is the bias signal for unit j of layer i
- W^i is the weight matrix for layer i

2.5.2 Restricted Boltzmann Machine

Boltzmann machines (BMs) is bidirectional connected networks of stochastic processing units [29]. It is two layer energy based model which models visible layer variable distribution using hidden units as shown in the figure 2.4. Restricted Boltzmann Machine is a

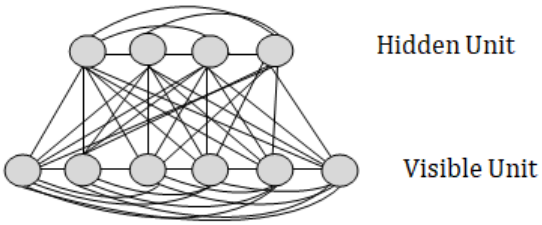


Figure 2.4: Boltzmann Machine

bipartite graph in which visible units that represent observations are connected to binary, stochastic hidden units using undirected weighted connections [30]. They are restricted in the sense that there are no visible-visible or hidden-hidden connections as shown in the figure 2.5.

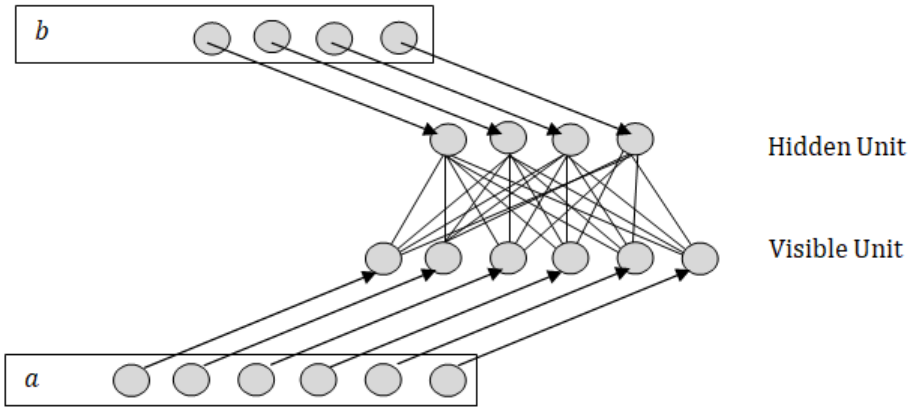


Figure 2.5: Restricted Boltzmann Machine

where:

- a is visible layer bias
- b is hidden layer bias

RBMs have an efficient training procedure which makes them suitable as building blocks for DBN.The training set can be model using two layers in which the visible unit connected to the input vector and the feature extractor correspond to the hidden units of RBM [29]. The energy of joint configuration (v, h) of visible and hidden units expressed in equation (2.4).

$$E(v,h,\theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H} W_{ij}v_i h_j - \sum_{i=1}^{V} b_i v_i - \sum_{j=1}^{H} a_j h_j \tag{2.4}$$

where

- v_i, h_j are the state of visible unit i and hidden unit j respectively
- a_j, b_i are the bias of visible unit i and hidden unit j respectively
- w_{ij} is the weight of the connection between visible unit i and hidden unit j

The joint probability of the visible and hidden unit of RBM is a function of the above energy function and given in equation (2.5).

$$p(v,h) = \frac{1}{Z}e^{-E(v,h)} \tag{2.5}$$

where:

- Z is the partition function and given by summing over all possible pairs of visible and hidden vectors as expressed in equation (2.6)

$$Z = \sum_{v,h} e^{-E(v,h)} \tag{2.6}$$

The probability that the network assigns to a visible vector, v, is given by summing over all possible hidden vectors as expressed in equation (2.7).

$$p(v) = \frac{1}{Z}\sum_{h} e^{-E(v,h)} \tag{2.7}$$

Gibbs Markov chain sampling of the visible and hidden unit pair of variables used to estimate the gradient on the log-likelihood of RBM [29]. It is the process of consecutive

sampling of hidden unit (h) given visible unit (v) then visible unit (v) given hidden unit (h) until the end of the chain. The chain starts from t=0 meaning from the input vector then proceed to (v_t,h_t) where t is the number of sampling iteration. The derivative of the log-probability of a training vector with respect to a weight is given in equation (2.8).

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.8}$$

The angle brackets are used to denote expectations under the distribution specified by the subscript that follows. This leads to a very simple learning rule for performing stochastic steepest descent in the log probability of the training data as shown in equation (2.9)

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{2.9}$$

Where:

- ϵ is learning rate of the weight update.

The above equation state that the weight updates is the difference of expectation of visible and hidden unit udder data and model distribution.

Since there is no direct connection between hidden units of RBM, getting the unbiased sample $\epsilon(\langle v_i h_j \rangle_{data})$ is easy from the training vector. The state of each hidden unit h_j of the set to 1 given the training vector can computed using probability equation (2.10).

$$p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{2.10}$$

Also to get the sample of each visible unit v_i of the visible layer given the hidden layer can computed in the following probability equation (2.11).

$$p(v_i = 1|h) = \sigma(a_j + \sum_j h_j w_{ij}) \tag{2.11}$$

But getting an unbiased sample of $\langle v_i h_j \rangle_{model}$ is not as simple as the data distribution. It could be calculated using Constructive Divergence (CD) which is approximation of the gradient through alternative Gibbs sampling starting from visible layer units for some specified time [29]. It is denoted as CD_n where n denotes number of full alternative Gibbs sampling. A single iteration of alternating Gibbs sampling consists of updating all of the hidden units in parallel using equation (2.10) followed by updating all of the

visible units in parallel using equation (2.11).

The fast way of training as proposed by [29] include setting the state of the visible units to the training vector, then sample the hidden units in parallel from the visible units using equation (2.10). The final step is the reconstruction process which is sampling the visible units in parallel from the hidden unit using equation (2.11). Then the weight update equation (2.9) can be simplified as shown in equation (2.12), (2.13), and (2.14).

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}) \tag{2.12}$$

$$\Delta a = \epsilon(\langle v_i \rangle_{data} - \langle v_i \rangle_{model}) \tag{2.13}$$

$$\Delta b = \epsilon(\langle h_j \rangle_{data} - \langle h_j \rangle_{model}) \tag{2.14}$$

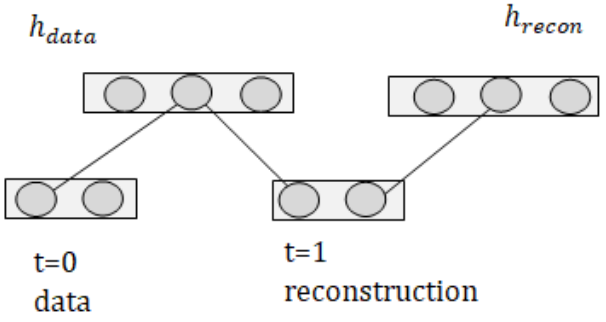


Figure 2.6: Constructive divergence with n=1

In classification problem like wireless IDS the dataset include input vector X that feed to the machine learning algorithm and target class Y which is the expected result of machine learning algorithm from the input dataset. Since DBN is constructed from stuck of RBMs the last RBM is discriminative RBM [31], which models the join distribution of input vector and target class as shown in the figure 2.7.

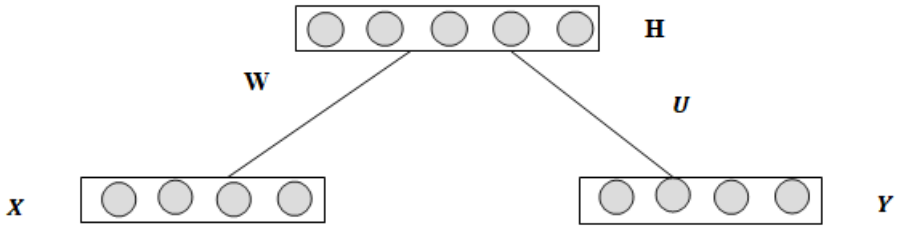


Figure 2.7: Discriminative Restricted Boltzmann Machine

where:

- X is inpute vector from which RBM models hidden layer units
- Y is RBM target output
- H is hidden layer
- W weighted connection between input vector visible units and hidden units
- U weighted connection between target class units and hidden units

The energy function is expressed as follow in equation (2.15).

$$E(v,l,h,\theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H} W_{ij}v_i h_j - \sum_{y=1}^{L}\sum_{j=1}^{H} W_{yj}h_j l_y - \sum_{i=1}^{V} a_i v_i - \sum_{j=1}^{H} b_j h_j - \sum_{y=1}^{L} c_y l_y \quad (2.15)$$

where:

- V is visible layer
- H is hidden layer
- L is target class layer
- v is visible unit of visible layer
- h is hidden unit of hidden layer
- l is target class unit of target class layer
- a,b,c are bias signal for visible, hidden and target class layers respectively
- $\theta = (W, b, c, d, U)$

The conditional probability of target class given hidden layer units computed as follows in equation (2.16).

$$p(l_y = 1|h) = softmax(\sum_{j=1}^{H} w_{yj}h_j + c_y) \quad (2.16)$$

The weight update role between hidden and target class follows the same question (2.12), (2.13) and (2.14).

2.5.3 Greedy Layer-Wise Training of Deep Belief Networks

According to [27, 28], DBN is stack of RBM and can be trained in unsupervised greedy manner one layer at a time for pre-training. It is the way of training deep architecture by training each part separately and stacking them to from the full structure. For DBN training Greedy layer-wise training is the way of fast training for DBN in unsupervised manner for pre-training. By training each layer of DBN in sequential way and feeding lower layer output to upper layer as input and initializing upper layer weight from transpose of lower layer weight, it is possible to pre train DBN as shown in figure 2.8. Since each layer of DBN is composed of RBM, training each layer of DBN is equivalent to training of respective RBM. It results better optimization of a network than traditional stochastic gradient descent training. The algorithm of greedy layer-wise unsupervised training for a DBN can be generalized as following

1. The input vector for model building is $h(0) = X$ to the first RBM of DBN.
2. Use the extracted data from the first layer as an input data to the second layer. This extracted data can either mean activation data $p(h(1) = 1|h(0))$ or a set of samples of $p(h(1)|h(0))$.
3. Then train the second layer as RBM and keep the mean activation or sample from first layer as training data of the visible layer in this RBM.
4. Repeat step 2 and step 3 for all number of layers and for each step feed upwards either the mean activations or the samples.
5. Finally, adapt fine-tuning on all parameters of the unsupervised network and training by gradient descent on a supervised training criterion.

It is unsupervised training but can be applied to labeled data by learning a model that generates both the label and the data [28]. It is confirmed to bring a better generalization by initializing a local minimum (or local criterion) that helps to formulate a representation of high-level abstractions of the input to the network.

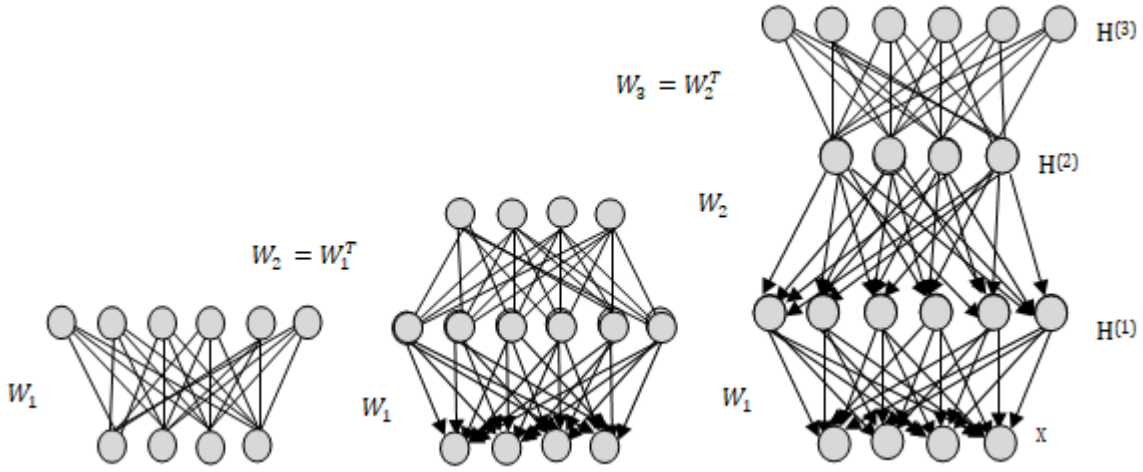


Figure 2.8: Greed layer-wise training procedure

2.5.4 Supervised Fine-tuning

Supervised fine-tuning is considered as the second step in DBN training process [27, 28]. It is the way of tuning the parameters of the pre-trained network for better performance of the system. It uses the same training dataset and network structure with supervised machine learning algorithms.The most usual supervised algorithm which is used for DBN fine-tuning is BPNN.

Back Propagation Training Algorithm for Multi-layer Neural Network

Neural network is one of nature inspired modern computational algorithm. It was proposed based on the working principle of natural neural cell and their connection between them to form nerves system [32]. Neural cell is composed of nucleus where electro-chemical reaction takes please, dendrites that used to connect to other neural cell axon for synapses input, and axon that expend to connect other neuron cell dendrites as output of synapses. The strength of the axon is a matter of the level of signal received by the next neurons for electro-chemical reaction. Then in this way number of them connects to form nerves system.

So in modern computational science based on the above principle ANN is design for

computational system. The main building block of the proposed ANN was neuron as of natural nerves system. The artificial neuron contain nucleus where the computation takes please, several input signal, one output signal, and connection weight as shown in the figure 2.9.

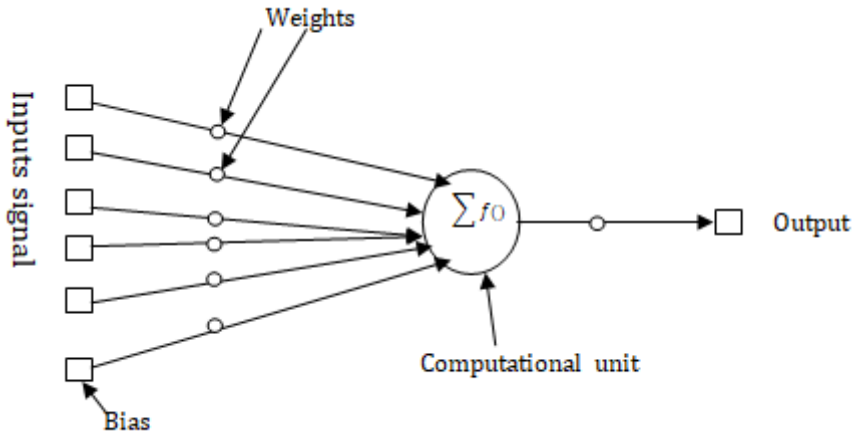


Figure 2.9: Single artificial neuron

As shown in the figure 2.9, the neuron receive input and bias signal with their weighted connection, then it compute the signal using activation function, and finally it let out the result as output signal. The input signal may come from one of the two sources; one is from the training dataset and the other is from the output of other previous layer neurons. The bias signal is additional signal to the input signal for the computational unit to remove zero input signals. The connections between neurons have weight which is the capability of amplifying or attenuating the signal that pass through it. Then the computational unit using the weighted input signal and the activation function it compute the result.

The activation function is the nonlinear differentiable mathematical formula to compute the output of the specified neurons output using the weighted input signals [33]. Its final value is bonded between two values. There are number of mathematical formulas that can be used as activation function of neuron which are sigmoid, hyperbolic tangent, hard limiting threshold, etc. and as example lets see sigmoid function. First lets consider a training dataset with X input having N features and Y output having M classes. Then the sigmoid activation function formula and its derivative are described in equation (2.17)

and (2.18) respectively.

$$f(y) = \frac{1}{1 + \epsilon^{-y}} \tag{2.17}$$

$$\frac{df(y)}{dy} = f(y)(1 - f(y)) \tag{2.18}$$

$$y = \sum_{i=1}^{N} x_i w_i \tag{2.19}$$

where:

- x_i is i^{th} feature input
- w_i is i^{th} feature input connection weight

The formula shows that the activation function is inversely proportional with the summation of the product of the input signal and the respective connection weight.

In ANN collection of neurons which are in parallel and does not communicate with each other forms layer of the network. The simplest form of ANN has two layers. The first layer is the input layer that represents only the input data and it does no computation at all. The next layer is the output layer. It contains number of computational neurons to compute the output from the input data, bias signal and weight of the respective signals. The number of neuron in the input layer is most of the time equal to the input dataset features and in the output layer equal to the number of classes in the case of classification application.

The other and the more advanced form of ANN is multi-layer neural network; which contain at list one hidden layer other than input and output layer [33, 34]. The number of hidden layer ranges from one to infinity and most of the time it is determined based on the design principle of the researchers. Also the number of neurons in each hidden layers ranges from zero to infinities. In the figure 2.10, three layer neural network graphical structure is depicted assuming I input signal, J hidden layer neurons, and K output layer neurons.

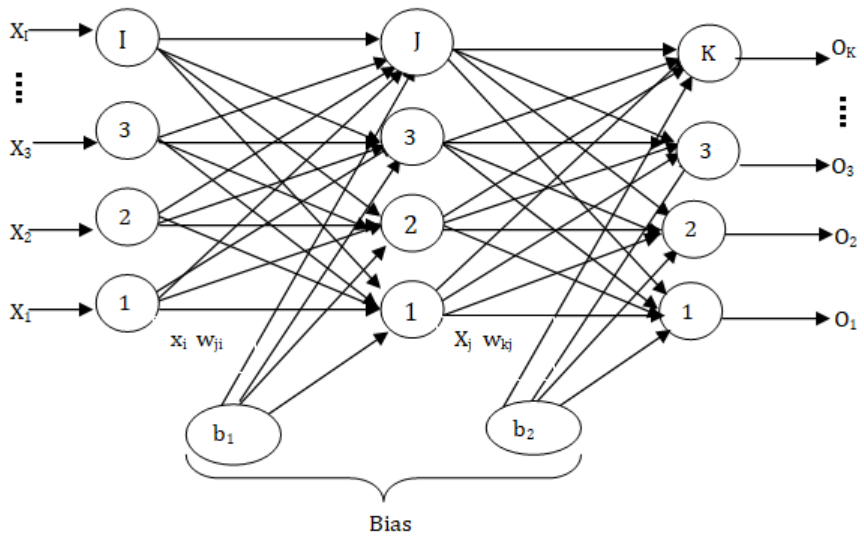


Figure 2.10: Three layer feed forward neural network

where

- x_i is the input signal
- w_{ji} and w_{kj} it weight matrix between the input layer and hidden layer and hidden layer and output layer
- b_1 and b_2 are bias signals for hidden and output layers respectively

Until now the general structure of ANN is discussed starting from its inspiration to how network of neurons can formed. Then the next main concept is how to train such network to make is artificially intelligent for the given task. So here is the place feed forward back propagation come in to application. It is the way of training the network using the training dataset and propagating back the error of the computational result [33,34]. The feed forward back propagation has two phases to train the given neural network. The first phase is feed forwarding of the input signal through each layer until output layer and the second phase is back propagating the error between the desired and the resulting out put back to each layer until before reaching the input layer. In the first step the output of any neuron in respective layer computed using equation (2.20) in combination with the activation function.

$$x_j = \sigma(\sum_{i=1}^{I} x_i w_{ji} + b_i) \tag{2.20}$$

where:

- σ is the activation function

The result of one layer will feed to the next layer until the output layer, and when it reach the output layer it will be end of feed forward computation and it became the beginning of the next back propagation process.

The back propagation algorithm is the method of training multi-layer neural networks using the gradient optimization method. The basic element of this algorithm is the energy function that defined as a quadratic sum of the difference between the actual output signals and the desired values as shown in equation (2.21).

$$E = \frac{1}{2}\sum_{i=1}^{p}\sum_{k=1}^{m}({y_k}^{(i)} - d_k^{(i)})^2 \tag{2.21}$$

where:

- E is the energy function which is the square of the error between the output signal and the desired signal.
- p is number of training vector.
- m is number of classes or output layer neurons.

So to reduce the above energy function value through training, back propagation follows the following steps.

1. Apply the actual input signal vector X
   (a) calculate the output signal under each hidden and output layer using equation (2.20).
   (b) Calculate the gradient of activation function in each neuron of each layer using the derivative of the activation function as expressed in equation (2.18)

   .

2. Create the back propagation network by reversing the direction of signal transmission
   (a) Replace the activation function by its derivative.

(b) The input vector at former output layer and the current input layer is the error between the actual and the desired value.

(c) The weight modification proceeds on the bases of the result in one feed forward and back ward propagation using equation (2.24).

3. Repeat one and two for all training samples as much time until the stopping criteria of the algorithm is reached.

The weight modification in each training steps can be computed using the following equation.

$$w_{ji}(t+1) = w_{ji}(t) - \epsilon \nabla E(w) \tag{2.22}$$

where:

- w_{ji} is weight of connection from neuron i to j
- $\nabla E(w)$ is the gradient of the energy function
- ϵ is training coefficient
- $t+1$ is next training time and t is current training time

The above formula states that the current weight update is the difference between the previous weight and training coefficient multiplied by the gradient of the energy function. The previous weight is obviously known, but the gradient of the energy function obtained by differentiating the energy function with respect to respective weight of neurons as follows.

$$\nabla E(w) = (o - t) * x_{ji} \frac{df(y)}{dy} \tag{2.23}$$

where:

- $(o - t)$ is the error of the current training step
- x_{ji} is the signal of the current connection which is under consideration to modify its weight
- $\frac{df(y)}{dy}$ is the derivative of the activation function

Back propagation algorithm in its training process some time it may stack on the local minima while it try to reduce the energy function [33]. So there should be some additive values to take off from the local minima which are function of the previous weight of

the specified connection known as momentum. So the final formula for weight update of connection between two neuron of consecutive layer is computed using the following formula.

$$w_{ji}(t+1) = w_{ji}(t) - \epsilon(o-y) * x_{ji}\frac{df(y)}{dy} + \alpha\Delta w_{ji}(t) \tag{2.24}$$

$$\Delta w_{ji}(t) = w_{ji}(t) - w_{ji}(t-1) \tag{2.25}$$

where:

- t is current execution time, $(t+1)$ is next execution time and $(t-1)$ is previous execution time
- o is desired output
- y is current system output
- $\frac{df(y)}{dy}$ is the gradient of the activation function
- α is momentum coefficient

Chapter 3

System Design and Implementation

In chapter 2, we discussed about theoretical background on some concepts and knowledges of the thesis including wireless LAN standards, structure, security, vulnerability and threats, literature reviews on wired and wireless IDS, and DBN algorithm with its training methodologies. Now on in this chapter the system design and implementation are discussed in detail. The design includes modeling the system, designing the preprocessing mechanism, discussing the design perspective of DBN and specifying parameters of the system and performance metrics. The implementation part discusses the implementation environment and the implementation methodology of the preprocessing mechanism and the classification system using pseudo code.

The proposed system is IDS for wireless network using DBN algorithm. The system starts from preprocessing of AWID dataset, selecting features from the preprocessed dataset, then training the system using full featured and selected feature dataset for IDS classification and testing the classification performance of the system for both full featured and selected feature dataset.

3.1 System design

The proposed system architecture is shown in the figure 3.1. It starts by selecting specific dataset for training and testing from the AWID dataset collection. Then the next step is preprocessing of both the training and testing datasets and the result is normalized version of both training and testing datasets. The preprocessed dataset is able to use for training and testing purpose of the system. But before that there are two steps that are able to produce other two datasets. The first one is feature selection process to select the most discriminant features of the datasets. The second process is preparing K-fold cross validation data. Finally from the above steps there are three datasets which are full featured dataset, list of selected feature datasets, and K-fold cross validation dataset.

Then the next step is the training of DBN using each of the datasets. Training DBN and testing its performance is main part of the system because it is the one that used to classify the wireless LAN traffic to its normal and attack classes. The training process contains two steps, that are unsupervised pre-training using greedy layer wise training through stacked RBM and fine-tuning the weight parameter of pre-trained DBN using BPNN algorithm. Then the results of the above two training process is trained DBN classifier.

Now on the system is ready for testing and performance evaluation. The testing process is similar to the training step in the case of dataset application; meaning the DBN that is trained with full feature dataset will be tested with the full feature dataset test data and in same way DBN trained with selected feature datasets will be tested using the selected feature test datasets of the testing dataset. Also in the case of cross validation for each fold the system is evaluated using the validation data and the average of all fold accuracy is reported as cross validation accuracy. The design process of each step will be discussed in the upcoming subsection in details.

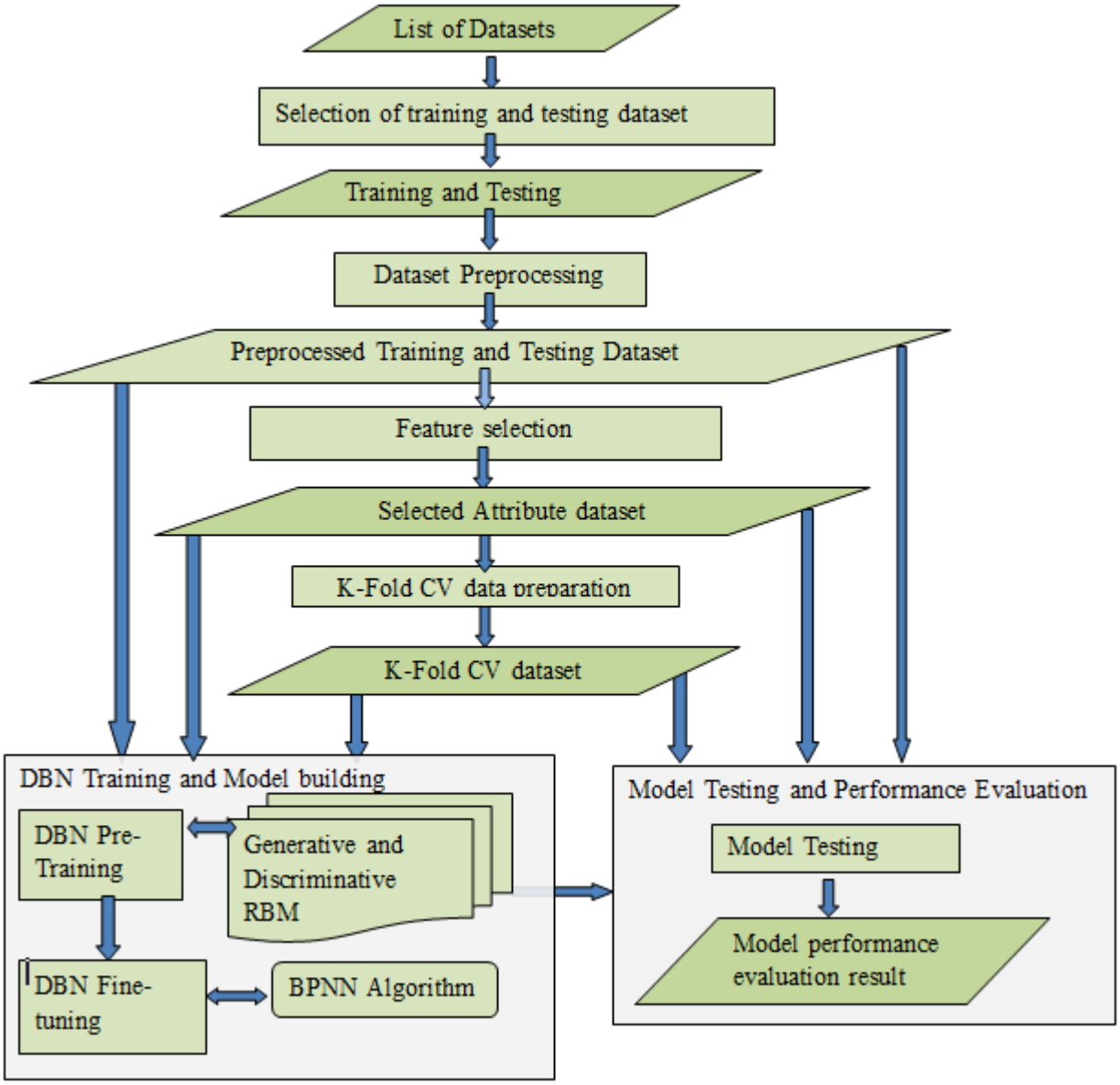


Figure 3.1: System work flow

3.1.1 Preprocessing and Feature selection Design

The purpose of the preprocessing step is to prepare the dataset in the way that usable for machine learning classification algorithms. Because the proposed DBN algorithm only works on numeric valued input and having range in the same scale between zero and one. So before taking any dataset as input for DBN algorithm based classification system the dataset should be prepared in such a way that usable for the system first.

As discussed in the section 2.4, AWID dataset is collection of a number of different datasets. So the first step in dataset preprocessing is selection of the specific size and class distribution from the dataset collection for both training and testing processo of the system. The selected dataset for training and testing purpose is the high label, reduced version of AWID dataset collection namely AWID-CLS-R-Trn for training and AWID-CLS-R-Tst for testing as recommended by [3] for researches that are based on limited resource and singe node computation due to its smaller size relative to the other datasets. The selected training and testing dataset properties are described in tabular form as show in the table 3.1

Table 3.1: File structure of the selected AWID

File Name	**Hrs**	**Tot_Recs**	**Nor_Recs**	**Att_Recs**	**Size_MB**
AWID-CLS-R-Trn	1	1795575	1633190	162385	935
AWID-CLS-R-Tst	1/3	575643	530785	44858	297

Where:

- **Hrs**: is the number of hours invested in monitoring the network to produce the dataset
- **Tot_Recs**: is total record of the file
- **Nor_Recs**: is the normal traffic record in the file
- **Att_Recs**: is the attack traffic record in the file
- **Size_MB**: the size of the file in MB

The attacks and normal class distribution in both the training and testing part of the dataset is shown in table 3.2a and table 3.2b respectively. As shown in the two tables, from the total number of the data in the dataset normal class is above 91% and the attacks class divide the rest percentage almost proportionally.

Table 3.2: Class data distribution in the training and testing dataset

(a) Training dataset

Class	**Data size**
Flooding	48484
Impersonation	48522
Injection	65379
Normal	1633190
Total	**1795575**

(b) Testing dataset

Class	**Data size**
Flooding	8097
Impersonation	20079
Injection	16682
Normal	530785
Total	**575643**

AWID dataset is combination of different data type based on the nature of each attributes. The data types in the dataset are decimal numeric, hexadecimal for MAC address, and nominal string values for SSID values and attack class types [3]. Also the scale of each attributes in the dataset is unbalanced and some attribute values are not specified or they are missing values.

So it is clear that there should be some mechanisms which preprocess the dataset to the more appropriate form. The preprocessing mechanism has the following steps.

- In the dataset there are some values which are unassigned or missing values represented as question mark (?), so there should be methodology to assign numeric values for such missed values of attributes. The methodology that followed in this thesis is assigning 0 as value for those not available values.
- The second part is string value encoding to numeric values. The SSID nominal value attack is encoded to numeric value based on their frequency in the specified attribute of the dataset. The other string valued attribute in the dataset is the attack class values and its encoding to numeric values is done as shown in table 3.3.

Table 3.3: Numeric encoding of attack classes

No	Class	Numeric Encoding
1	Flooding	1
2	Impersonation	2
3	Injection	3
4	Normal	4

- The third step is type casting of hexadecimal values to their decimal equivalent. This involves converting hexadecimal values to their respective numeric value.
- The fourth step in preprocessing is scaling all the attributes between zero and one using equation (3.1).

$$Z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.1}$$

  where:

  - Z_i is scaled value
  - x_i is current attribute value from the dataset
  - min(x) is minimum value in the given attribute and
  - max(x) maximum value in the given attribute
- The final step is removing NaN valued attributes from the dataset because the in the next stage of the system only real number valued attributes are important.

The final output of this preprocessing step will be normalized and numeric valued version of the selected dataset.

The next step in preprocessing is the feature selection process. In this process there are two steps. The first one is removing the redundant attributes from the preprocessed dataset by keeping single one of each kind. The second process is feature selection using the entropy of each attributes for the class labeling. It is done using Weka implemented IGR feature selection algorithm. Based on their IGR value attribute are ranked and by

setting up some threshold values the most discriminant attributes are selected. To show the relation between number of selected attributes in the dataset and the performance of the system three thresholds are used that result three different new datasets. The thresholds are selected using intuitive guess in their increasing order from smallest to the largest and the number of attributes in each resulting datasets decreases. The resulting datasets are called using their number of attributes.

After all the above steps there are five version of dataset for system modeling and performance evaluation. The dataset that gives the best performance measure gives the best feature set for intrusion detection using the designed and the implemented system.

3.1.2 Cross Validation

Cross validation is an evaluation method for classification algorithm by splitting the given dataset in to two parts that are the training part and testing part [35, 36]. But maximizing both the training and testing dataset is the main tradeoff issue, because maximizing training dataset means best system modeling and maximizing the testing dataset result the best system validation. To solve these issues there are four commonly used cross validation approaches that are listed as follow.

- **Hold out cross validation:-** in which the dataset is divided in to two that are training and testing datasets. Then the prediction on the test dataset will reported as accuracy of the model. It is cost effective but it is prone to large variation.
- **K-fold cross validation:-** it randomly divide the dataset in to K roughly equal size parts then it train by using K-1 dataset and evaluate with rest one dataset and the process continues K times by rotating the training and evaluation parts as shown in the figure 3.2. Then the average accuracy of the k-fold model will reported as accuracy of the model. It is more expensive than hold out cross validation but it is the best way of training and validating the model.

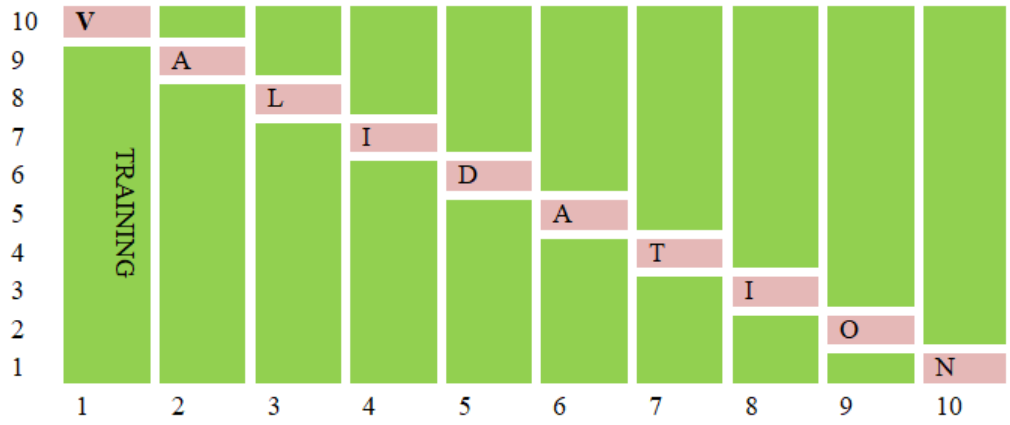


Figure 3.2: 10-fold cross validation simulation

- **Leave one out cross validation**:- is a spatial kind of K-fold cross validation where K equals to the number of data points in the dataset. It is the most efficient way of modeling and validating the system but in resource perspective it is the most expensive technique.
- **Bootstrap cross validation**:- it takes a random sample with replacement from the training set B times. Since the sampling is with replacement, there is a very strong likelihood that some training set samples will be represented more than once. As a consequence of this, some training set data points will not be contained in the bootstrap sample. The model is trained on the bootstrap sample and those data points not in that sample are predicted as hold-outs.

The main idea of using cross validation in this thesis is to maximize both system modeling and evaluation. The original dataset have separate training and testing dataset in the form of holdout cross validation. So K-fold CV is used as the other variation of system validation in addition to the original hold out form datasets. In K-fold cross validation both the training and the testing dataset are merged before splitting in to K-folds rather than using only the training dataset to increase the number of data points and to let the system learn large variation. For this thesis 10 is selected as the values of K so the validation process known as 10-fold cross validation.

3.1.3 Classification System Design and Implementation

The main process of the system is to train DBN classifier based on the given dataset and to classify as correctly as possible the test dataset using its training experience. The classification system architecture come from the architecture of DBN as shown in the figure 3.3. It has input layer that directly connected to the input dataset, number of hidden layers which model the input data sequentially up to the output layer, and the output layer which result the classification value of the system. Each of the layers from input through hidden up to the output layer they contains number computing units. The number of computing units varies from layer to layer. The number of computing units in the input layer is equal to the number of input attributes in the dataset and also in the case of output layer it is equal to the number of classes in the dataset.

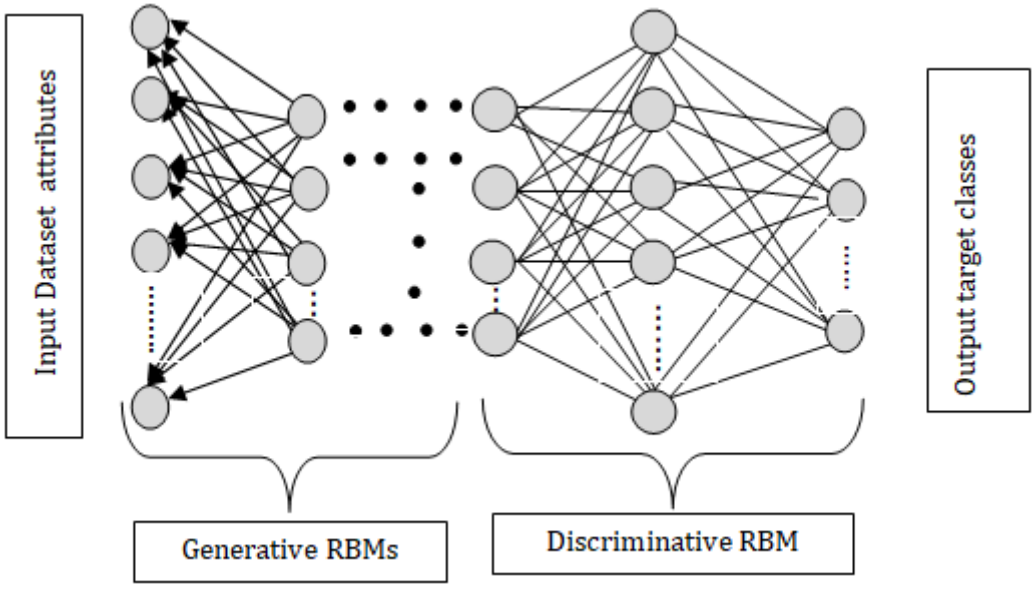


Figure 3.3: Deep Belief Networks architecture

Deep Belief Networks Training

As discussed in section 2.5.1, DBN is multilayer generative model which try to model the input data through deep hierarchical architecture. The training algorithm that follows to train the classifier DBN is Greedy layer-wise unsupervised for pre-training and BPNN for fine-tuning as shown in figure 3.4.

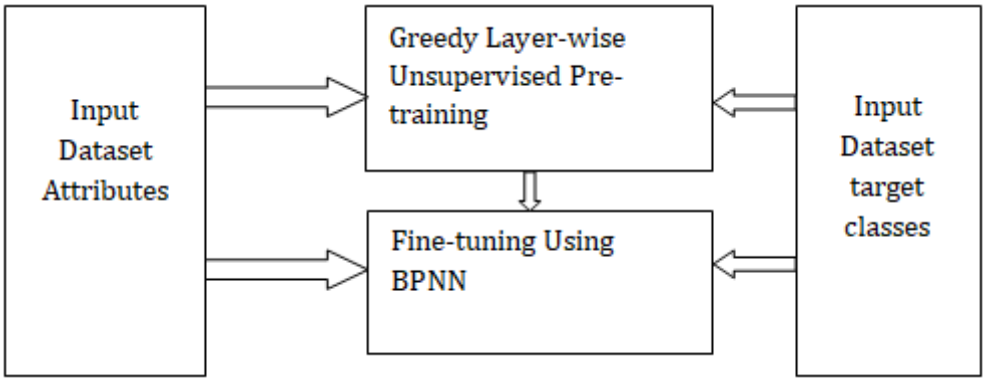


Figure 3.4: Deep Belief Networks training

The unsupervised pre-training method is used to initialize the hidden layer weights while building deeper model. In the pre-training step of DBN each layer of the network will train using RBM in greedy layer-wise manner as shown in Figure 2.8. The trained RBM will be used to construct the pre-trained layer of DBN. Starting from the first up to the last layer they will modeled as generative RBM but the last layer of DBN with the classifier layer will modeled as discriminative RBM as shown in the figure 3.3. The training process will continue from the input layer up to the last layer in greedy layer-wise manner for unsupervised pre-training of DBN.

The last but not the list part of the system training part is supervised fine-tuning. The purpose of fine-tuning is to more refine the hidden layer weight value using supervised algorithms. In this thesis BPNN will used for fine tuning the pre-trained DBN. It will apply on the same architecture, meaning the number of layers and their respective computing units are the same with the previous pre-trained structure.

Restricted Boltzmann Training

As discussed in section 2.5.2, RBM is two layer generative models. It tries to model the training dataset over its hidden layer units. RBM used to train DBN in unsupervised greedy layer wise manner.

The training rules that are specified in equation (2.12) for weight update, (2.13) for visible bias update and (2.14) for hidden bias update will be used for training RBMs in DBN. For sampling the visible units from the hidden units and visible biases equation (2.10) will be used and for sampling the hidden units from the visible units and hidden biases equation (2.11) will be used. To collect the statistics for update rules the real valued probability of both visible and hidden unit will used instead of stochastic binary states to reduce sampling noise for faster learning as recommended by [29] for CD_1 where 1 is the number of full alternative Gibbs sampling in CD.

In RBM training parameters play a very significant rule on its performance. So it is very important to take under consideration while designing RBM training algorithm.

Training epoch: - it is the total training steps of the machine learning algorithm using the specified dataset. For each single epoch the RBM will compute CD_n through consecutive Gibbs samplings, then it update its weight and bias values and finally it calculate the reconstruction error statistics. Its value extends in the range of positive integer and the specific value for this thesis will be determined using preliminary experiment in the next chapter.

Learning rate(ϵ):- as shown in the training rules equation (2.12), (2.13), and (2.14) learning rate is the controlling parameter for weight and bias update values. Having large learning rate mean the weight change in each iteration is large then finally the weight will explode and the system may over fit earlier than it was expected also having a very small learning rate will result to long the learning process [29]. The value range is between zero and one and recommended value are 0.05, 0.1, and 0.3 in literature [37] and specifically for CD 0.1 was recommended by [27] and in this thesis it will determined using preliminary experiment in the next chapter.

Momentum: - momentum simply adds a fraction of the previous weight update to the current one. The momentum parameter is used to prevent the system from converging to a local minimum or saddle point [29]. A high momentum parameter can also help

to increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local minima, and can also slow down the training of the system. The value of momentum ranges in [0, 1] typical value 0.9 is used in this thesis as recommended by [29].

Number computing unit hidden layer:- as discussed in section 2.5.2, hidden layer is feature extraction part of RBM. For such purpose it use a number of computing units. Its range is extended in positive integer. So to decide the specific number of units in the hidden layer preliminary experiment will be undertaken and the results will discuss in the next chapter.

Weight and bias initialization: - in machine learning weight mean the tuning mechanism of the connection between computing units between two consecutive layers. So the algorithm learns the pattern from the training dataset by adjusting its weight parameter through its learning epoch. Also the bias is used as initialization signal for computing units of their respective layers. since DBN is stack of RBMs; the first RBM weight will initialized using small random values that have zero mean Gaussian distribution with standard deviation of about 0.01 using equation (3.2) as recommended by [29] and the rest upper RBMs weight will initialized to the transpose of their respective lower RBM weight as shown in figure 2.8. Both the visible and the hidden biases will initialize to zero as recommended by [29].

$$w = 0.01 * randn(NV_{units}, NH_{units}) \tag{3.2}$$

where:

- NV_{units} is number of visible units
- NH_{units} is number of hidden units

Weight decay:- through training epoch weight decay is important to control over fitting of the model [29]. There are two version of weight decay, the first one is absolute value decay (L1) that push a lot of the weights to be exactly zero while allowing some to grow large and the other one is square value (L2) which tends to drive all the weights to smaller

values [38]. Among them L2 is used in this thesis. The recommended value of L2 by [29] ranges from 0.01 to 0.00001 and 0.00001 is used in this thesis.

Bach size: - is the number of training instance per batch. The typical value depends on the training data. The recommended value by [29] is in range of [10 100] and 100 will be used as batch size in all experiments.

Number of Gibbs Sampling (CD) :- In this thesis number of full alternative Gibbs sampling n is chosen to be 1 because of resource constraint and it is enough for experimental implementation of RBM accourding to [29].

Back Propagation Neural Network

Back propagation neural network will used for fine-tuning the pre-trained DBN. After unsupervised pre-training the DBN will transform to feed forward multilayer neural network. The layer of multilayer neural network with its parameter derived from the DBN layers. Then back propagation training algorithm is used to train the network for fine-tuning the weight of DBN using equation (2.24) and (2.25).

Performance Metrics

Finally after designing the system and implementation there should be some performance evaluation mechanism. They are the means of measuring the performance of the system based on the output of the system. There are number of classification task evaluation mechanism as stated by different literatures [39–42] and each of them are used to measures the different aspect of performance of the given system.

Based on the number of classes in the system output classification system can grouped in to two groups. Classifier with two classes is known as binary classifier while classifier more than two classes is known as multi class classifier [39]. The proposed wireless IDS system is multiclass classifier, because the dataset contains three attack classes and one normal class totally four classes then the performance metrics consideration will focus of

multiclass classification.

Monitoring training progress: - there should be some quantitative mechanism to monitor the progress of system training to see how it is going through training process. For RBM training the reconstruction error is used to monitor the training progress as recommended by [29]. It is the squared error of the difference of data and the final CD stage reconstruction of visible units. For back propagation mean square error between the expected and the predicted value will be used.

Training time: - it is the total time to train the system using the training dataset.

Confusion matrix:- it is matrix of the classification result that used to show the performance of the system. It is used as the base of other performace measurements. It has tabular structure as shown in table 3.4, by considering the proposed AWID dataset classes.

Table 3.4: Confusion matrix

	Actual				
Predicted		***Normal***	***Injection***	***Flooding***	***Impersonation***
	Normal	TP	F	F	F
	Injection	F	TP	F	F
	Flooding	F	F	TP	F
	Impersonation	F	F	F	TP

As shown in the table 3.4, it contains truly classified positive and negative value and falsely classified positive and negative values. The diagonal is a truly classified value and the rest is falsely classified values. where:

- ***True Positive (TP)***:- is the amount of the specified class detected when it is actually that class.

- ***True Negative (TN)***:- it is the sum of all true positives except the specified class true positive value.
- ***False Positive or False alarm (FP)***:- It is the sum of predicted class row except TP value of that class. It is the amount of data predicted as that class but actually they are not member of predicted class.
- ***False Negative (FN)***:- It is the sum of actual class column except TP value of that class.It is the amount of actual class members but predicted as member of other class.

Accuracy: it is the percentage of truly classified value. It can calculate as the diagonal of confusion matrix divide by the total number (Num) of sample in the dataset using equation (3.3).

$$accuracy = \frac{TP + TN}{Num} \tag{3.3}$$

Precision: - is the percentage of the specified class TP out of that class predicted positive value and can calculate using equation (3.4).

$$precision = \frac{TP}{TP + FP} \tag{3.4}$$

Specificity: - is the percentage of the specified class TN out of that class predicted negative value and can calculate using equation (3.5).

$$Specificity = \frac{TN}{TN + FP} \tag{3.5}$$

Recall: -is the percentage of the specified class TP out of that class actual positive value and can calculate using equation (3.6).

$$recall = \frac{TP}{TP + FN} \tag{3.6}$$

F-Measure:- it is the weighted harmonic mean of the specified class precision and recall and can calculate using equation (3.7).

$$F = \frac{(1 + \beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{3.7}$$

Where:

- β is integer value to favuor recall over precision according to [40] and in this thesis value of β is 1.

So in the research result and discussion the above performance matrices will be used as implemented wireless IDS performance measuring dependent variable. The independent variables are described as follows.

- The complexity of the implemented algorithms.
- The data size of the selected AWID dataset family.
- The number of attribute in the dataset.

So in this thesis the different combination of the above independent variables will be experimented to study their effect on the system performance metrics.

3.2 Implementation

After designing the system the next step is the implementation of the system on certain environment to model the proposed system through training and experiment the performance of the system. The implementation has two parts. The first is setting up implementation and testing environment. The second is the programming approach to implement the system.

3.2.1 Implementation and Experimental Environment

The implementation environment can be seen in two perspectives that are hardware and software. In software perspective for implementation Matlab programing language will be used with Matlab R2016a version tool. For attribute selection Weka version 3.8 will be used. The operating system is Window 10 pro 64-bit. In hardware perspective Dell OptiPlex 7020 desktop with following specification is used for debugging the implementation code, training in the process of model building and testing the performance of the system.

- Intel(R) Core(TM) i5-5500U CPU @ 3.70GHz

- 8.0 GB DDR3 memory (RAM)

3.2.2 System Implementation

The system is implemented using concept of Object Oriented Programming (OOP) approach on Matlab programming language. The main power of OOP is its ability to abstract concepts in the way of real world objects. In OOP Object is an entity with properties and operations. Properties and operation are encapsulated with common template for similar objects called as class. Next lets discuss each algorithm implementation using their respective pseudo codes.

Deep Belief Networks Implementation

The pseudo code of DBN is shown in algorithm 1. It starts from initializing the number of layers in DBN and model of DBN [27, 28]. Then for each layer of DBN it add specific RBM object. If the layer of DBN is above the first input layer is initialize the layers RBM weight by its previous layers RBM weight transpose. Then it checks for specific layer RBM is discriminative or generative [31]. If is discriminative it train in discriminative way or if it is generative RBM it train in generative way. Finally it fine tunes the parameters by using BPNN algorithm.

Algorithm 1 Deep Belief Networks algorithm

Initialize num_layers, DBN_models

/* pre-train DBN */

for all itL in num_layers **do**

$Dbn\_rbm\{itL\} = RBM(DBN\_model\{itL\})$

if $itL >= 2$ **then**

$Dbn\_rbm\{itL\}.weight = Dbn\_rbm\{itL-1\}.weight$

end if

if $((dbn\_rbm\{itL\}.classify = true) or (itL == layers))$ **then**

$dbn\_rbm\{itL\}.train(data, target)$ //train discriminative way

else

$dbn\_rbm\{itL\}.train(data)$ //train generative way

end if

*/*get the current RBM output for next RBM input */*

$data = dbn\_rbm\{itL\}.hidd\_expectetion(data)$

end for

/ fine tune pre-trained DBN using back propagation */*

for all itL IN num_layers **do**

$back_propitL = Dbn_rbmitL$

end for

$back\_prop\{itL\}.train(data, target)$

Restricted Boltzmann Machine Implementation

As discussed in section 3.1.3, RBM is used for DBN unsupervised pre-training purpose. The RBM training implementation pseudo code is shown in algorithm 2. It starts from initializing the parameters [27] [29,30]. Then it goes to the individual iteration of training epoch and first it extracts the hidden units from the input data, then it constructs the visible units using the hidden layer units, finally it extract the hidden unit from the reconstructed hidden units. Finally using the above consecutive feature extraction and construction results it update the weight of the connection and respective visible and hidden bias.

Algorithm 2 Restricted Boltzmann Machine algorithm

Initialize parameters

for all *ep* in *epoch* **do**

for all *itH* in *num_hidd_units* **do**

Compute *hid_exp_data* using equation (2.10)

end for

for all *itV* in *num_vis_units* **do**

Compute *vis_exp_data* using equation (2.11)

end for

for all *itH* in num_hidd_units **do**

Compute *hid_exp_data* using equation (2.10)

end for

update weight using equation (2.22)

update hidden bias using equation (2.14)

update visible bias using equation (2.13)

end for

Back Propagation Neural Network Implementation

In the proposed system training process, as discussed in section 3.1.3, the final step is fine-tuning the pre-trained DBN parameters using BPNN supervised machine learning algorithm [27]. As shown in algorithm 1, the BPNN starts by initializing its parameter from pre-trained DBN. Then for each epoch it forward the input signal from the input layer to the output layer and then it back propagate the error back to the first hidden layer [33, 34]. In the feed forward step it computes the activation function and its derivative of each units of each layer up to the output layer and finally it compute the error between the predicted and the desired values. In the error back propagation step it propagates the error back to each layer of neurons until the first hidden layer and it update the parameters.

Algorithm 3 Back Propagation Neural Network algorithm

```
Initialize parameters
for all ep in epoch do
  /* feed forward */
  for all itL in num_layers do
    for all itN in num_node{itL} do
      /* using equation (2.20) */
      compute the activation of the node
    end for
  end for
  /* Propagate the errors backward through the network */
  for all itN in num_node{num_layers} do
    /* using equation (2.21) */
    calculate the square of the error signal
  end for
  for all itL in num_layers − 1 do
    for all itN in num_node{itL} do
      /* using equation (2.24) and (2.25) */
      update the weight
    end for
  end for
end for
```

Chapter 4

Experimental Result and Discussion

In chapter 3, the system design and implementation was discussed in detail. In this chapter the designed and implemented system will be trained to create the proposed model and tested for its performance using performance metrics that are discussed in section 3.1.3 and the result will be discussed in detail. The experimental procedure starts from preprocessing the dataset, then next feature selection experiment will undertake to select the most discriminative features of the dataset for the proposed system, then the preliminary experiment will undertake on some of system parameters and finally the system is trained using the training dataset and it will tested for its performance.

As discussed in the design section 3.1.3, there are different system parameters that should be tuned to improve the system performance. Setting their value is the critical thing while experimenting on the proposed system. Some of them were decided by reviewing previous literatures in section 3.1.3 and for the rest of them there should be preliminary experiment and their values are discussed in this chapter. So after preparing the dataset before experimenting the system performance preliminary experiment is undertaken to decide those parameter values.

After all the above experiments the system is ready for training and testing. In this

section there is number of experiments and discussion to analyze the system performance form different perspectives. The first experiment is testing the effect of training dataset size on the performance of the system. This is done by training the system with different percentage of the training dataset starting from twenty percent up to hundred percent with twenty percent increment and testing the performance of the trained system with the test dataset. The second experiment is on the effect of attribute selection on the performance of the system. The attribute set that resulted best performance is selected as the most discriminant attribute set of the proposed system. Then the last experiment is on 10-fold cross validation for better system training and evaluation. The cross validation data is prepared using the previous experiment best performer attribute set.

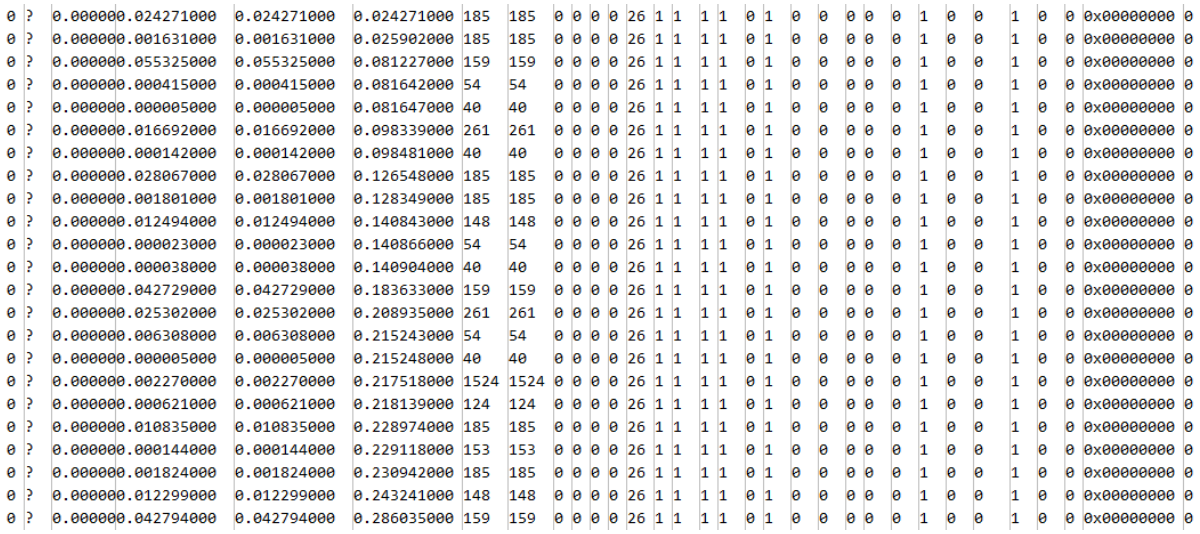
4.1 Preprocessing and Feature Selection Experiment and Result Discussion

The preprocessing experiment is done using the design principle as discussed in section 3.1.1. The first step was replacing the missing values of attributes by zero. Then next step was converting all non-numeric value of attributes to their numeric equivalent using the specified methodology and finally the numeric value of each attributes was normalized to be between zero and one.

The preprocessing is done on both the training and the testing dataset. The sample dataset before preprocessing is shown in figure 4.1.

After the preprocessing experiment out of the total features 52 of them has value of NaN because of attributes minimum and maximum value is equal. So the normalization formula (3.1) gives divide by zero that results NaN value. Since machine learning algorithms only compute on numeric values then all NaN valued attributes was removed from the dataset before moving to the next step. The resulting dataset of the preprocessing experiment is 102 features which are numeric valued and normalized between zero and one and it is called as version one dataset [1]. Now on the dataset is ready to use for

[1]Dataset version means nothing but it is the way of identifying different datasets resulted from preprocessing and attribute selection procedures.



0	?	0.000000	0.024271000	0.024271000	0.024271000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.001631000	0.001631000	0.025902000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.055325000	0.055325000	0.081227000	159	159	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000415000	0.000415000	0.081642000	54	54	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000005000	0.000005000	0.081647000	40	40	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.016692000	0.016692000	0.098339000	261	261	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000142000	0.000142000	0.098481000	40	40	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.028067000	0.028067000	0.126548000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.001801000	0.001801000	0.128349000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.012494000	0.012494000	0.140843000	148	148	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000023000	0.000023000	0.140866000	54	54	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000038000	0.000038000	0.140904000	40	40	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.042729000	0.042729000	0.183633000	159	159	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.025302000	0.025302000	0.208935000	261	261	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.006308000	0.006308000	0.215243000	54	54	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000005000	0.000005000	0.215248000	40	40	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.002270000	0.002270000	0.217518000	1524	1524	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000621000	0.000621000	0.218139000	124	124	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.010835000	0.010835000	0.228974000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.000144000	0.000144000	0.229118000	153	153	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.001824000	0.001824000	0.230942000	185	185	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.012299000	0.012299000	0.243241000	148	148	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0
0	?	0.000000	0.042794000	0.042794000	0.286035000	159	159	0	0	0	0	26	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0x00000000	0

Figure 4.1: Sample dataset before preprocessing

machine learning computation. The sample from the preprocessed dataset is shown in figure 4.2.

0.7273	1	0	1	0	0	0	0	1	0.0217	2.9299e...	2.9299...	0.1593	0.1265	0.1593	0	0.7983
0.1818	0	0.6154	0	0	0	0	0	0	0	1	1	0.1593	0.1593	0.1593	0	0.4642
0.9091	1	0.6154	1	0	0	0	0	1	0.0030	0.7504	0.7504	0.1593	2.9299e-04	0.1593	0	0.2996
0.9091	1	0.6154	1	0	0	0	0	1	0.0030	0.7504	0.7504	0.1593	2.9299e-04	0.1593	0	0.3084
0.6591	0.5000	1	0	0	0	0	0	0	0	0.1593	0	0	0	0	0	0
0.1818	0	0.6154	0	0	0	0	0	0	0	1	1	0.2193	0.2193	0.2193	0	0.3321
0.9091	1	0.6154	1	0	0	0	0	1	0	0.7504	0.7504	0.1593	2.9299e-04	0.1593	0	0.0989
0.6591	0.5000	1	0	0	0	0	0	0	0	5.7584e...	0	0	0	0	0	0
0.7273	1	0	1	0	0	1	0	1	0	1	1	0.1593	3.7012e-04	0.1593	0	0.0899
0.6591	0.5000	1	0	0	0	0	0	0	0	0.7504	0	0	0	0	0	0
0.9091	1	0.6154	1	0	1	0	0	1	0.0030	0.2358	0.2358	0.1593	2.9299e-04	0.1593	0	0.2593
0.7273	1	0	0.5000	0	0	0	0	1	0.0217	0.1593	1	5.7584...	5.7584e-04	0.1593	0	0.2066
0.2727	0	0.9231	0	0	1	0	0	0	0.0217	5.7584e...	5.7584...	0.1593	0.1593	0.1593	0	0.9839
0.9091	1	0.6154	1	0	0	0	0	1	0.0030	3.7945e...	3.7945...	0.1593	2.9299e-04	0.1593	0	0.2589
0.9091	1	0.6154	1	0	0	0	0	1	0.0030	0.7504	0.7504	0.1593	2.9299e-04	0.1593	0	0.6168
0.6591	0.5000	1	0	0	0	0	0	0	0	5.7584e...	0	0	0	0	0	0
0.2727	0	0.9231	0	0	1	0	0	0	0.0217	5.7584e...	5.7584...	0.1593	0.1593	0.1593	0	0.8029
0.2727	0	0.9231	0	0	1	0	0	0	0.0217	0.2510	0.2510	0.1593	0.1593	0.1593	0	0.5201
0.9091	1	0.6154	1	0	0	0	0	1	0.0030	0.7504	0.7504	0.1593	2.9299e-04	0.1593	0	0.5231

Figure 4.2: Preprocessed sample dataset

The next step after preprocessing the dataset is feature selection and there are two consecutive steps as discussed in section 3.1.1. The first process is removing the redundant attributes by keeping only a single one of them. It results 68 unique features, excluding the class labeling, and it is called as second version dataset. The second process is implementing IGR feature selection algorithm on the result of the first process. As discussed in section 3.1.1, three thresholds are set to select attributes which result three dataset versions. The first version is using 0.015 threshold value and the resulting dataset has 41

attributes, the second version is using 0.05 and the resulting dataset has 34 attributes, and the last version is results 25 attribute dataset with 0.2 thresholds. So the three new datasets are called as version three, four and five respectively. The IGR computation result of 68 attributes is shown in the graph 4.3.

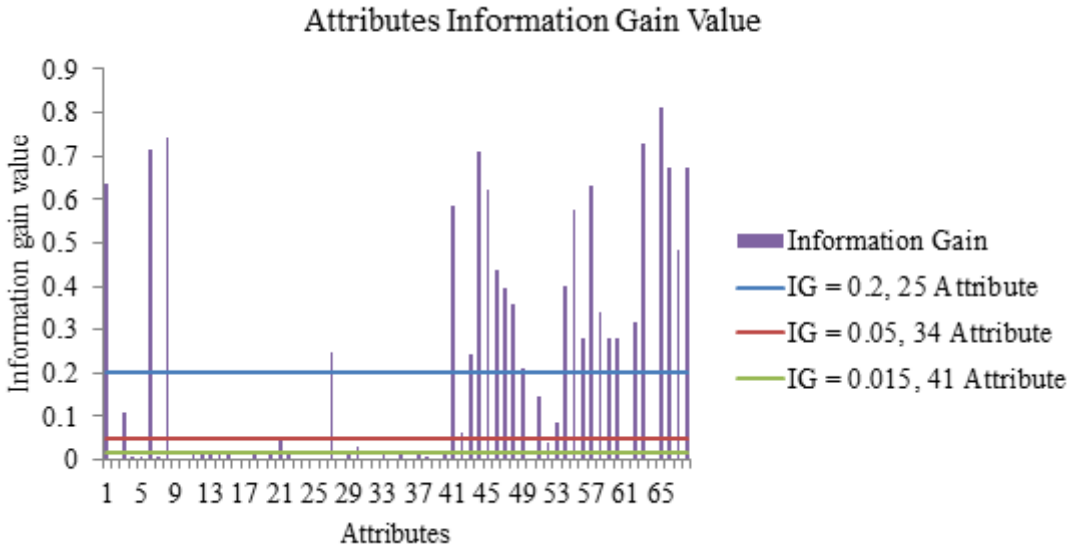


Figure 4.3: Attributes information gain ratio value

4.2 Experimental Setup

As discussed in section 3.1.3, specifying some of system parameters and their values scientifically based on experimental procedure is critical to the designed system model performance. A value of some of the parameters was able to determine by reviewing related researches but for the rest of attributes it important to undertake some preliminary experiments. Here three parameters, namely RBM training epoch, RBM learning rate, number of computing units in RBM hidden, are experimented and its value is specified. In those experiments while testing the specified parameter the other parameter value is set to either previously explored value or random value is selected from its value range for temporary use until experimenting on it.

4.2.1 Number of Epoch for RBM Training

To decide on the number of training iteration of RBM the following experiment is under taken. The range of epoch for this experiment is between 1 up to 100. So the experiment is running on a single RBM for 100 epochs and the reconstruction error is recorded as shown in the graph 4.4. It shows that the reconstruction error is dropping smoothly up to around 50 training iterations, then after that it continuous constant and even it increase a little bit which shows that the occurrences of over fitting. So based on the above result 50 is selected as the number of epoch for model construction in this thesis.

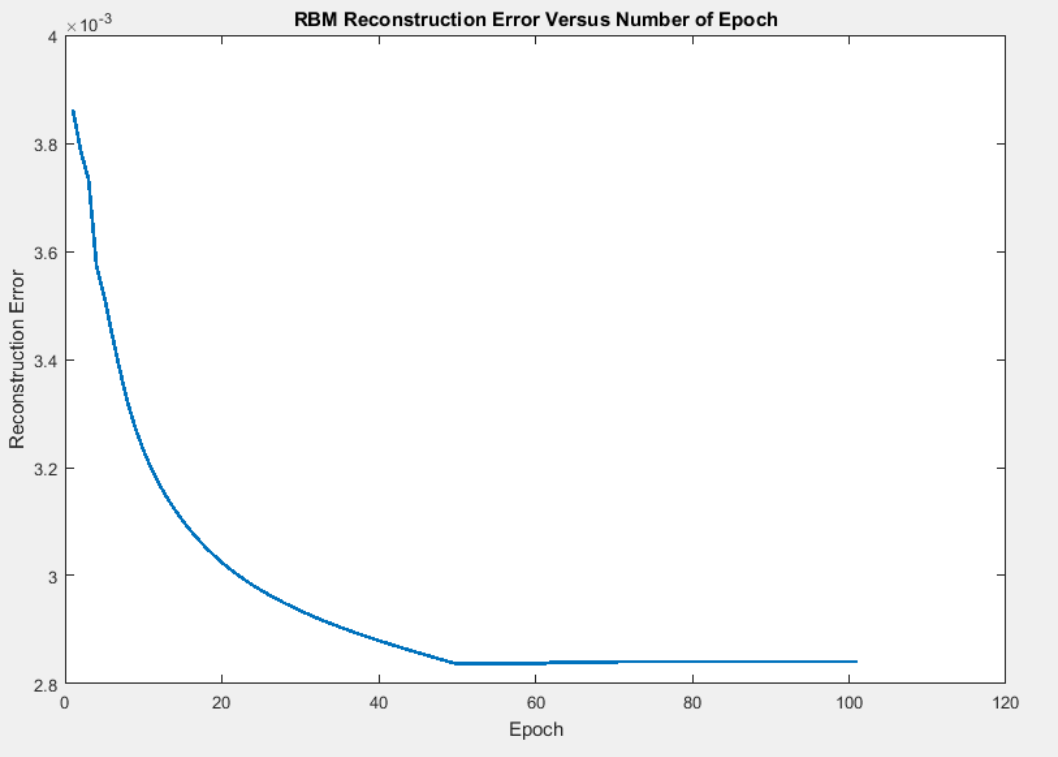


Figure 4.4: The effect of number of epoch on the training performance of the system

4.2.2 Learning Rate for RBM Training

To decide the learning rate of RBM the following experiment was undertaken. As recommended by [27,29,37], the learning rate range for this experiment is between 0.05, 0.1, and 0.3.For each value in the specified range the RBM is trained and its reconstruction errors are recorded as shown in the graph 4.5. It shows that RBM trained with 0.1 learning rate fails bellow the other two graphs and it means that it results smaller reconstruction

error than learning rate with 0.05 and 0.3. Since small reconstruction error shows that the fast convergence and better model building process of the system 0.1 is selected as learning rate.

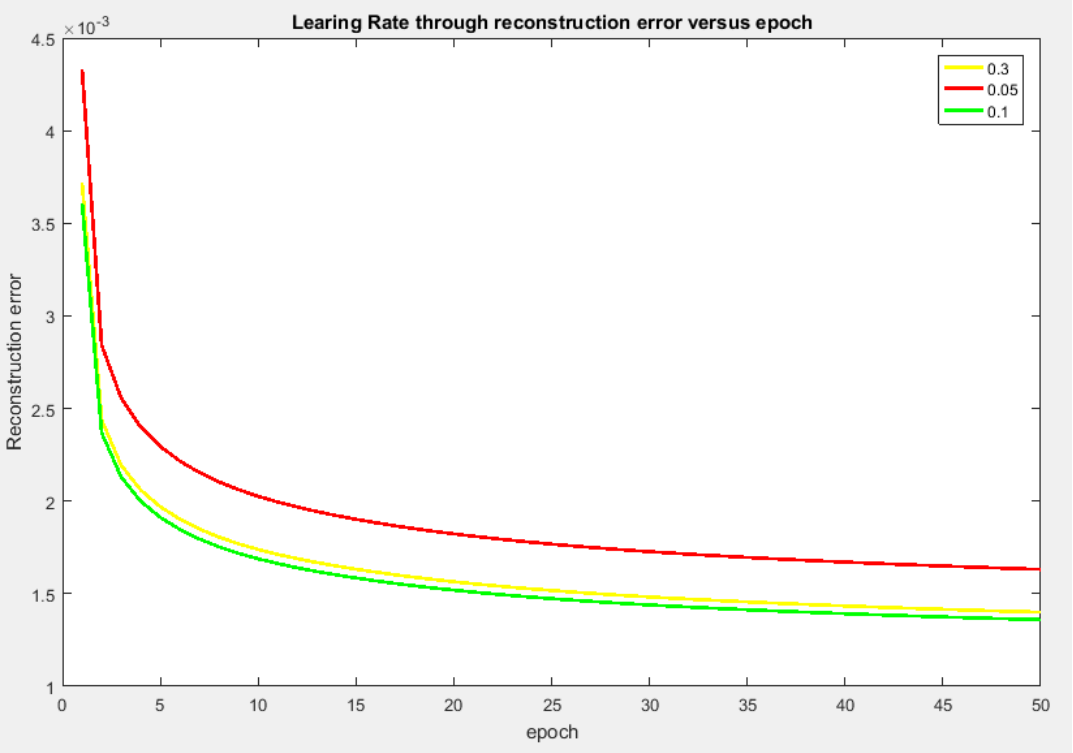


Figure 4.5: The effect of learning rate on the training performance of the system

4.2.3 Number of Computing Units in RBM Hidden Layer

To decide the number of computing units in the hidden layer of RBM the following experiment is undertaken. Since it is not possible to experiment on all possible positive integer computing units first it is wise to guess the most probable range of values in initiative way. So the range of computing units for experiment is 35, 65, 150, and 250. The performance measure that used to decide in the number of computing units is reconstruction error of the RBM model and the result is shown in the graph 4.6. It shows that RBM with 65 hidden layer computing units fall bellows the other three graphs, so 65 numbers of computing units are used in the hidden layer of RBM.

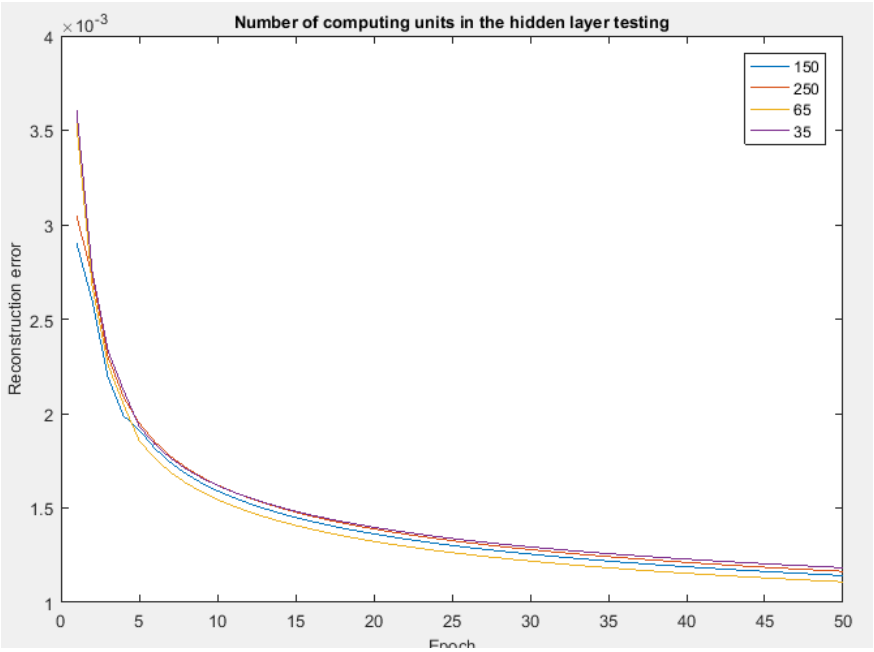


Figure 4.6: The effect of number on computing unit of the training performance of the system

4.3 System Experiment and Result Discussion

After preprocessing the dataset and deciding the system parameters the next step is training the system to create model using training dataset and evaluation of model performance using testing dataset. Since there are three datasets which are the result of preprocessing, feature selection and 10-crosst validation dataset preparation the system is evaluated independently with each version of the dataset. So in this section there are three independent experiments that are based on each datasets. In each experiment the system trained with the training dataset then the trained system is tested with the corresponding testing dataset for its performance.

The result from the 102 feature dataset is used as the base line to compare with the selected feature result and 10-fold cross validation result. The result from selected feature experiment is compared with the 102 feature experiment result to see the effect of feature selection on model performance. Also the result from 10-fold cross validation is compared to the selected feature result to show the power of cross validation on model building than

using the usual training and testing dataset.

4.3.1 Experiment on the Effect of Dataset Size on the Performance of the System

The purpose of this experiment is to see the performance progress of the system while the size of the dataset increases. The size of the dataset means how much percent of the dataset is used to train the system in model building process. The proposed percentages of the dataset for this experiment are listed as shown in table 4.1.

Table 4.1: Training dataset percentage

No	**Percentage**	**Numerical Size**
1	20	359115
2	40	718230
3	60	1077345
4	80	1436460
5	100	1795575

All percentage calculations are applied only to the training dataset but not on the testing dataset because the aim is just to see the effect of the size of the dataset on the performance of the system. So the system is trained using each percentage of the dataset and the trained system performance is evaluated using the testing dataset.

As shown in figure 4.7, the classification accuracy of the model increases while the dataset size is increasing. When the size is approximating to the full size of the training dataset, the accuracy difference between each consecutive training size is getting lower and lower. The best performance the system was able to achieve was 98.55% with 100% of the training dataset.

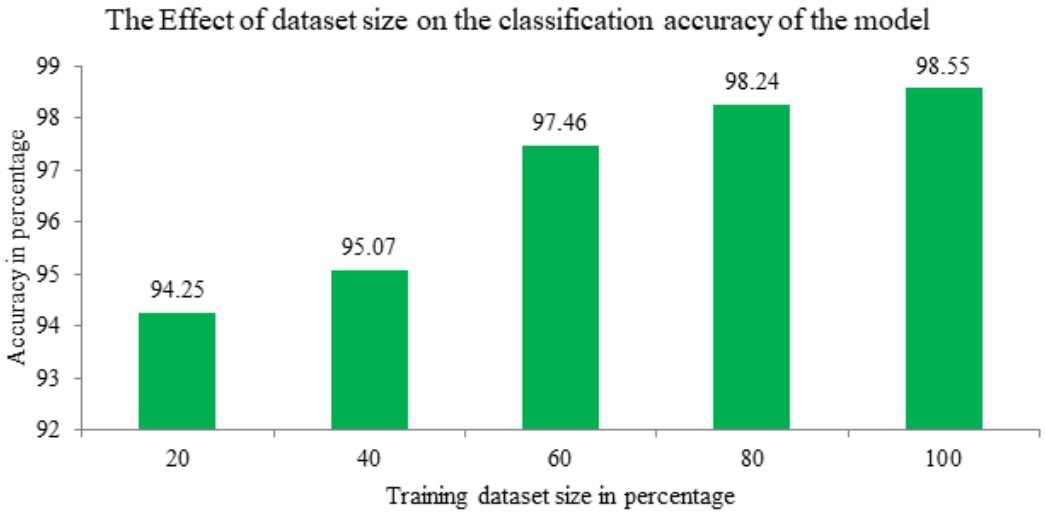


Figure 4.7: The effect of dataset size on the classification accuracy of the model

As discussed in section 3.1.3, classification accuracy is not the only way to measure the performance of classification model especially in the case of unbalanced multi-class dataset. So lets analyze and discuss each of the performance metrics in detail. As discussed in section 3.1.3, almost all performance metrics analysis is based on confusion matrix of the classification result. As discussed above the best result is achieved on 100% training dataset which is 98.55% and its confusion matrix is shown in table 4.2. It shows that the classification accuracy of the Normal and Injection class is close to 100% which is 99.97% and 99.94% respectively, Flooding is classified 75.17% correctly, and Impersonation is classified 69.23% correctly.

Table 4.2: Confusion matrix for 100% 102 attributes training dataset

.	**Normal**	**Injection**	**Flooding**	**Impersonation**
Normal	99.9717	0.0539	24.8240	30.7436
Injection	0.0034	99.9460	0	0
Flooding	0.0243	0	75.1760	0.0199
Impersonation	0.0006	0	0	69.2365

So based on the above confusion matrix other performance metrics are computed for each

classes and the result is shown in table 4.3.

Table 4.3: Performance measure for 102 attribute model

Classes	**Precision**	**specificity**	**Recall**	**F-measure**
Normal	98.47966045	81.7363	99.97174	99.22009
Injection	99.89215745	99.99673109	99.94605	99.9191
Flooding	97.86173633	99.97630682	75.17599	85.03178
Impersonation	99.97842503	99.99945789	69.23652	81.81497

Precision or sensitivity is the number of true positive prediction of the specified class out of total positive prediction. As shown in table 4.3 based on their precision value the attack classes descending order is Impersonation, Injection, Normal, and Flooding. Impersonation is is the most precise then the other because as shown in 4.2, only Normal class 0.0006% falsely classified as Impersonation class. But Flooding class is less precise than the other because as as shown in 4.2, most of other classes false negative is false positive of Flooding class relative to its true positive.

In other way specificity is the measure of true negative prediction of the class out of the total negative prediction. Out of all classes Impersonation has higher specificity, Injection has the second, Flooding has the third, and Normal has the fourth and the last specificity value.

Recall is the measure of the specified class true positive prediction out of that class actual true positive. From the four class domain their recall value order is Normal, Injection, Flooding, and Impersonation. It means that in Normal and Injection the model was able to predict truly above 99.9% to their respective class out of the given true class data and in the case of Flooding and Impersonation it is around 75.17% and 69.23% respectively.

F-Measure it is the way of combining the specified class precision and recall using weighted

harmonic mean. Injection has the highest F-measure value 99.91% and then Normal with 99.22%, Flooding with 85.03%, and Impersonation with 81.81% value follows.

As shown from the above result discussion, Impersonation is the most difficult to detect and Flooding takes the second and Injection and Normal classes are the most detectable classes using the proposed IDS. When we see the nature of attacks as discussed in section 2.4.1, Impersonation attack is a collection of Caffe Latte, Hirte, Honeypot, and Evil tween attacks. The First two attacks, which are Caffe Latte and Hirte, are AP less type of Keystream Retrieving Attack [43, 44]. It means that attackers can use isolated clients to attack the target network based on their connection history with the target network, which makes difficult to get those attacks footprint while sniffing the target network activities. The second two attacks, which are Honeypot and Evil tween, are type of Man-in-the-Middle Attack [45, 46]. Honeypot is a network created by attackers to attract target clients to connect with it for the next level attack. Evil tween is a special type of Honeypot which advertise existing ESSID to fool clients to connect with it instead of valid ESSID. What makes too difficult to detect in the proposed IDS is both network are first they dont considered as attack and second is it is impossible to get signature of these types of networks [3]. Also Flooding is DoS attack based on the management frame [3, 47]. It is simple and easy to implement and it is effective, because the management frame is not protected in general. As discussed in section 2.4.1, it contains 13 different attacks. Almost all the attacks are developed based on the normal activity of 802.11 management frame which is not susceptible by the recipient of the message as malicious activity. For example in the case of Deauthentication attack, the aggressor transmit Deauthentication message in unprotected way on the behalf of AP and the receiving clients drop the associated network and it restart the authentication process without any precondition. It contain also the large variation of attacks, which uses each of management frame messages, which could makes difficult to find similar signature between those attacks as a single attack for the proposed system. In the case of Injection attack, in all attack types the attacker inject a large number of small encrypted data frame to get useful information to perform key retrieval process [48–50], which makes the attack more detectable by the proposed IDS system.

4.3.2 Experiment on the Effect of Attribute Selection on the performance of the system

Then the next experiment is on the effect attribute selection on the performance of the system. As discussed in section 4.1, there are four versions of datasets that are produced from feature selection process on the original preprocessed dataset with 102 attributes. So in this section the system is trained independently and its performance is evaluated for each version of the dataset. The resulting classification accuracy of the model built by each version of the dataset is shown in the figure 4.8. As it shows, the classification accuracy of the system increase up to some point but it decrease again while the number of attributes in the dataset further decreases. The first accuracy improvement is the result of removing duplicated attributes from the dataset that increase the classification accuracy from 98.55% to 98.79%. Then the next consecutive experiments are based of the feature selection using their IGR values. When 41 top discriminative attributes are used it increases the accuracy to 98.86% and it continues increasing to 98.97% with 34 attributes. But when further decreasing the number of attributes to 25, the accuracy decrease to 98.74%.

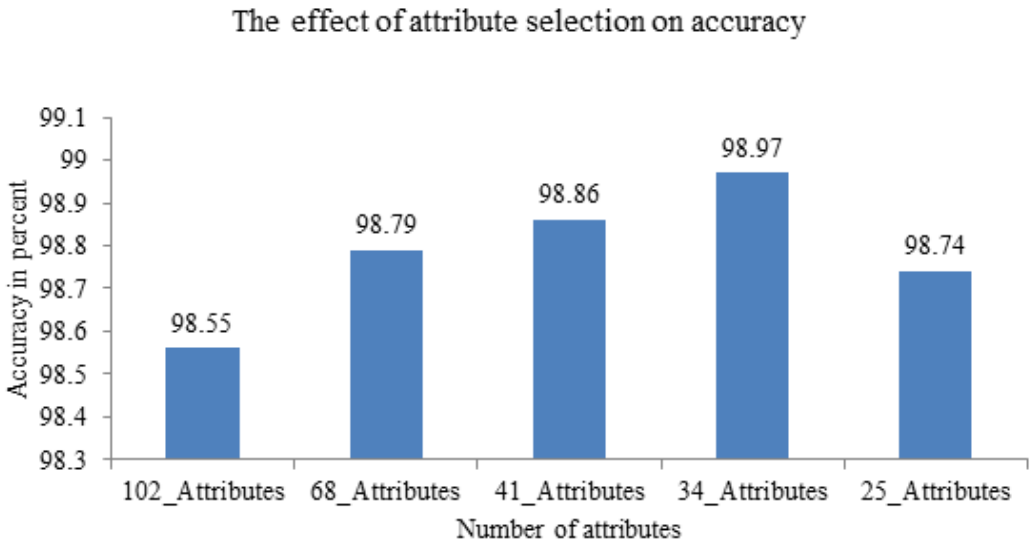


Figure 4.8: Classification accuracy for different attribute datasets

So based on the results, 34 attributes gives the best classification accuracy and those attributes are listed in table 4.4.

Table 4.4: The most discriminative attributes which result 98.97% classification accuracy

No	Attribute	No	Attribute
1	frame.time_epoch	18	wlan.duration
2	frame.time_delta	19	wlan.ra
3	frame.time_relative	20	wlan.da
4	frame.cap_len	21	wlan.ta
5	radiotap.flags.fcs	22	wlan.sa
6	radiotap.channel.freq	23	wlan.bssid
7	radiotap.channel.type.ofdm	24	wlan.frag
8	radiotap.channel.type.2ghz	25	wlan.seq
9	radiotap.antenna	26	wlan_mgt.fixed.timestamp
10	wlan.fc.type	27	wlan_mgt.fixed.beacon
11	wlan.fc.subtype	28	wlan_mgt.fixed.reason_code
12	wlan.fc.ds	29	wlan_mgt.ssid
13	wlan.fc.frag	30	wlan_mgt.tim.dtim_period
14	wlan.fc.retry	31	wlan.wep.iv
15	wlan.fc.pwrmgt	32	wlan.wep.icv
16	wlan.fc.moredata	33	wlan.qos.priority
17	wlan.fc.order	34	data.len

The other measurement is the effect of feature reduction on the execution cost of the system specifically the training time while models are built using the different attribute datasets. The resulting training time in second for each attribute set is shown in the figure 4.9.

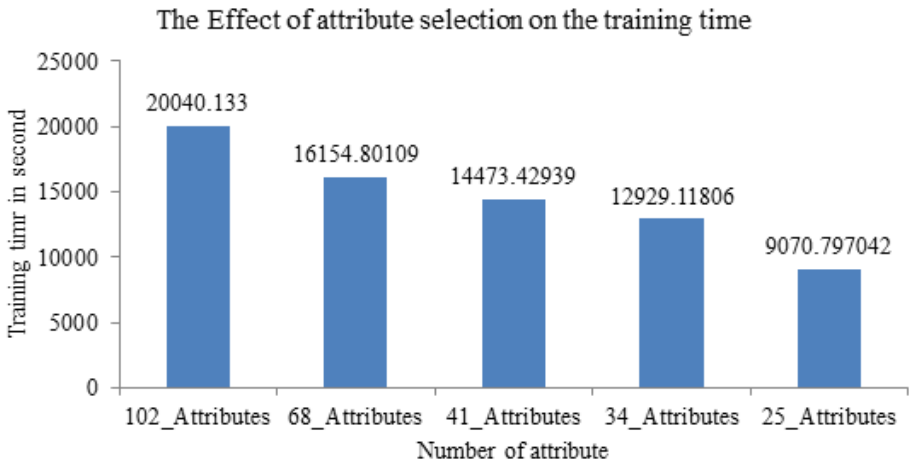


Figure 4.9: The effect of attribute selection on the training time

As shown in the figure 4.9, the training time decreases in proportion with the number of attributes in the dataset. With feature reduction while the system achieves the best classification accuracy it decreases the training time from five hour and thirty-four minutes to three hour and thirty-five minutes. This means the training time reduces around two hour while it scores the best classification accuracy.

As discussed in section 3.1.3, classsification accuracy is not the only measurement for performance of the system and in section 4.3.1 it is clearly shows that it is a good practice to make analysis on other performance metrics on the result of 34 attribute set dataset. So as usual first lets see the confusion matrix of the classification result as shown in the table 4.5. The table shows that, the system is able to classify the Normal class above 99.8%, Injection class 99.98%, Flooding class 93.49%, and Impersonation class 78.45% correctly. When the above result compared to the previous 102 attribute dataset result in the case of Normal and Injection classes classification accuracy is almost similar but in the case of Flooding and Impersonation classes there is a great improvement in their classification accuracy. For 102 attribute dataset Flooding is correctly classified 75.1% but in the case of 34 attribute dataset it is classified correctly 93.49% with improvement of 18.39%. Impersonation classification is also improved from 69.2% to 78.45% by 9.25%.

Precision, specificity, recall, and F-measure performance metrics are computed for each class based on the above confusion matrix as shown in the table 4.6. The table shows

Table 4.5: Confusion matrix for 34 feature dataset

.	**Normal**	**Injection**	**Flooding**	**Impersonation**
Normal	99.8046	0.0119	6.5042	21.4951
Injection	0.0038	99.988	0	0
Flooding	0.1650	0	93.4957	0.0548
Impersonation	0.0265	0	0	78.45012

that, Injection is the most precisely detected class and followed by Impersonation, Normal, and Flooding classes. In the case of specificity Injection is the most specific than the other class. The second specifically classified class is Impersonation followed by Flooding and the last specifically classified class is Normal class. Regarding recall Injection comes first and followed by Normal, Flooding, and Impersonation. In the case of F-measure value, which the combination of precision and recall, Injection class has the highest value and followed by Normal, Flooding, and Impersonation.

Table 4.6: Performance measure for 34 attribute model

Classes	**Precision**	**specificity**	**Recall**	**F-measure**
Normal	99.09389	89.19401	99.80463	99.44799
Injection	99.88017	99.99638	99.988	99.93406
Flooding	89.50047	99.84247	93.49573	91.45449
Impersonation	99.11282	99.97455	78.45012	87.57923

Finally it is clear that feature selection results an improvement on multiple performance metrics. It clearly increases the accuracy and reduces the training time. Even though it continues in reducing training time but after 34 attributes the classification accuracy declines down. So 34 feature set dataset is selected as the most discriminative attribute set for the system.

4.3.3 Experiment on 10-Fold Cross Validation and Result Discussion

As it is discussed in section 3.1.2, cross validation is a set of model building and performance evaluation methods and among them 10-fold cross validation is selected to be done here. Since the aim of 10-fold cross validation is better model building and at the same time better performance evaluation, the dataset is prepared using 34 features dataset. The system is trained with nine parts and its performance is tested using the rest one part out of the total ten folds. Then the process rotates through each part by switching the testing part each time. The experiment is undertaken for each round and the resulting classification accuracy of the 10-fold cross validation and their average is shown in the figure 4.10. As show in graph, the classification accuracy range from 99.96% to 99.983% with standard divination 0.01146 and average classification accuracy 99.9627%. When this result is compared with the above 34-attribute classification accuracy result, it is improved from 98.97% to 99.96% by around 0.99%. But the training time increase from three hour and thirty-five minutes to four hour and sixteen minutes for a single iteration and to train with total 10-fold cross validation data the training time is fourty-two hour and fourty-one minutes. So 10-fold cross validation is able to improve the system classification accuracy but it is clearly shown that it is expensive in computational time.

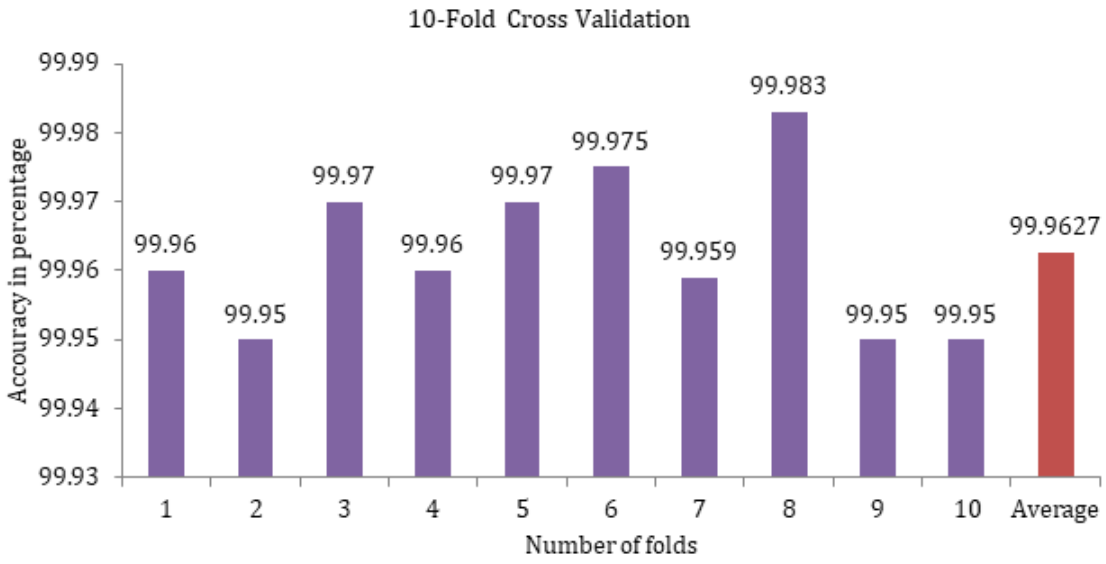


Figure 4.10: 10-Fold cross validation classification accuracy

Next lets see the other performance measures starting from the average confusion matrix of the the ten experiments.

Table 4.7: Confusion matrix for one of 10-Fold cross validations

.	**Normal**	**Injection**	**Flooding**	**Impersonation**
Normal	99.9713	0	0.0530	0.2478
Injection	0.0018	100	0	0
Flooding	0.0203	0	99.9469	0.0548
Impersonation	0.0064	0	0	99.7376

As shown in the table 4.7, the classification accuracy of each class is above 99.9% and their descending order is Injection, Normal, Flooding, and Impersonation. When we compare this result with previous 34 attribute dataset confusion matrix it improve the classification accuracy of all classes especially in the case of Flooding and Impersonation the improvement is more significant. The other performance metrics are also computed based on the this confusion matrix. As shown in the table 4.8, each class's classification has become more precise and specific when it is compared with the previous 34 attribute result.

Table 4.8: Performance measure for 10-fold cross validation

Classes	**Precision**	**specificity**	**Recall**	**F-measure**
Normal	99.99076	99.90349	99.97135	99.98105
Injection	99.93911	99.99782	100	99.96954
Flooding	99.22793	99.98099	99.94698	99.58616
Impersonation	99.7958	99.99392	99.73761	99.7667

4.4 Result Comparison with Previous Works on Wireless Intrusion Detection Systems

In this chapter, different experiments are undertaken on the implemented system to analyze the performance from different perspectives. In this section the result from those experiments are compared with different researches that are based on AWID dataset as discussed in section 2.3.2. The first research that evaluates AWID dataset using different machine learning algorithms is the research work by [3]. From their experiment they get different result using different machine learning algorithms and the best accuracy they were able to achieve was 96.1981% with J48, which is implementation of C4 algorithms. They have also selected twenty attributes manually and they were able to improve the classification accuracy to 96.2574%. The classification accuracy of Normal class was 99.99% using OneR, Injection was 99.98 using J48, Flooding was 72.69% using Naive Bayes, and Impersonation is 7.5% with random tree but they were able to improve Impersonation accuracy to 22% with twenty attribute with Nave Bayes.

The other work on wireless IDS using AWID dataset is the research work in [15] using different machine learning algorithms. They were able to achieve 95.12% using Random Tree with 41 attributes. The research in [16] was able to achieve 96.32% in both full attribute set and 20 attribute set dataset

The last but not the least work on the evaluation AWID dataset is on the improvement of Impersonation attack detection in the research work given in [17]. The researchers were able to achieve 65% classification accuracy on Impersonation attack with selected 35 attributes.

In this thesis it was able to achieve 98.55% with 102 features and was able to improve this result to 98.97% with 34 selected attributes. When we see the individual class classification accuracy Normal and Injection was able to classify above 99.9% in both 102 attributes and 34 attributes. In the case Flooding it was able to achieve 75.17% using 102 attributes dataset and it was able to improve to 93.45% using 34 attributes. Also

Impersonation is able to detect 69.236% when using 102 attributes and it was able improve to 78.45% using 34 attributes.

The other experimental result is using 10-fold cross validation and it is able to achieve average 99.9627% classification accuracy. The individual class classification accuracy is above 99.7%.

So the above result discussion clearly shows that the approach followed by this research is able to achieve better classification accuracy than similar previous works. Almost in all the above research works including this thesis, Normal and Injection classes were detected above 99.9%. But when we see the achievement on Flooding and Impersonation, the classification accuracy was poor and in this thesis the system can detect better than the work done by previous researches.

The other strength of this thesis is that it is implemented using single DBN algorithm as a single system. But in most of the researches they evaluate AWID dataset using different machine learning algorithms and each algorithm has different classification accuracy on each classes. So their works become difficult to implement as a single integrated efficient system but our system is able to improve all classes of attack detection accuracy and can be implemented using a single classification algorithm using only DBN.

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

In this thesis, wireless network IDS is proposed using DBN deep learning algorithm. For model building and performance evaluation, AWID dataset is used. Through literature review, we found that there is no research works that evaluate AWID dataset using DBN deep learning algorithm. Since DBN is proven for its ability in different classification and recognition tasks, the main objective of this thesis was to evaluate AWID dataset using DBN for performance improvement of wireless IDS.

In the proposed system, the training and testing dataset is collected form AWID dataset website [10] and in the preprocessing phase the dataset is normalized between zero and one before using for training and testing purpose. Finally NaN valued attributed are removed from the preprocessed dataset and this process reduce the attribute set from 154 to 102. Also attribute selection was implemented in two phases. The first one is removing redundant attributes from the dataset by keeping a single one of each it and in this process the attributes set is reduced to 68. The second phase in feature selection is done by applying IGR and using three thresholds three attribute set are selected for DBN training and testing. Totally after attribute selection process there are four versions of datasets, which are 68 attribute, 41 attribute, 34 attribute, and 25 attribute for attribute selection experiment.

The system is trained in two phase that are unsupervised pre-training and supervised fine-tuning of the pre-trained DBN. Since DBN is stack of RBM, RBM is used for unsupervised pre-training. Two kind of RBM are used in the training processes, which are generative RBM and discriminative RBM. The generative RBM is used for modeling DBN layers except the last layer which is modeled using discriminative RBM. For supervised fine tuning the weight of the pre-trained DBN back propagation multilayer neural network training is used.

A number of experiments have done on the implementation of the proposed system. The first Experiment is on the effect of the dataset size in the performance of the proposed system and the result proves that the classification accuracy of the system increases in proportion with the size of the training dataset and the best classification accuracy is achieved is 98.55% when using 100% of the training dataset. The second Experiment is on the effect of the attribute selection on the performance of the system. Each of the four versions of datasets are evaluated and the result shows that the classification accuracy increases until 34 attributes and it fall down when the attribute set farther reduced to 25. So in the proposed system feature selection was able to improve the classification accuracy to 98.97%. The third experiment is 10-fold cross validation for better model building and at the same time to better performance evaluation. Because it uses all data points from model training and testing by rotating each parts of the fold. The 10-fold data is prepared from 34 attribute dataset and experiment is done for each fold and the average classification accuracy becomes 99.9627%. The result clearly shows that using 10-fold cross validation it was able to improve the performance of the system

Finally the results from this thesis are compared with previous wireless IDS works that are based on AWID dataset. The proposed system was able to achieve better performance especially in the case of Flooding and Impersonation class detection.

5.2 Recommendation and Future Works

There are number of tasks that can be put as recommendations for the future work. The first task that can be done in the future is experimenting on the other versions of AWID datasets. Working on finer grained version would give better analysis on the detection performance of each single attacks and working on the full dataset would improve the performance of Flooding and Impersonation class detection.

The second task that can be recommended as future work is the implementation of the system for online application and testing other than offline as online implementation. Such kind of experiment gives an importance to analyze the performance of the system in live packet sniffing systems in real time environments. It is also easy to integrate reinforcement learning procedures to train the system about newly happening anomalies when the system operates in online mode.

References

[1] Alom, Md Zahangir, VenkataRamesh Bontupalli, and Tarek M. Taha. "Intrusion detection using DBN." *Aerospace and Electronics Conference (NAECON)*, 2015 National. IEEE, 2015.

[2] Ahmad, Javaid. "A Deep Learning Approach for Network Intrusion Detection System." *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. Vol. 35.

[3] Kolias, Constantinos, et al. "Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset." IEEE Communications Surveys & Tutorials 18.1 (2016): 184-208.

[4] Waliullah, Md, and Diane Gan. "Wireless LAN security threats & vulnerabilities." *International Journal of Advanced Computer Science and Applications* 5.1 (2014).

[5] Feng, Pan. "Wireless LAN security issues and solutions." *Robotics and Applications (ISRA), 2012 IEEE Symposium on*. IEEE, 2012.

[6] Parte, Smita, and Smriti Pandya. "Wireless LAN: Security Issues and Solutions." *Proceedings of International Conference on Innovation & Research in Technology for Sustainable Development (ICIRT 2012)*. Vol. 1. 2012.

[7] Salakhutdinov, Ruslan. *Learning deep generative models*. University of Toronto, 2009.

[8] Mohamed, Abdel-rahman, George Dahl, and Geoffrey Hinton. "Deep belief networks for phone recognition." *Nips workshop on deep learning for speech recognition and related applications.* Vol. 1. No. 9. 2009.

[9] NSL-KDD. http://nsl.cs.unb.ca/NSL-KDD/.(Accessed on 05/02/2017)

[10] AWID Dataset: http://icsdweb.aegean.gr/awid/index.html (Accessed on 05/02/2017)

[11] Arbaugh, William A. Real 802.11 security: Wi-Fi protected access and 802.11 i. Addison-Wesley Longman Publishing Co., Inc., 2003.

[12] Sheldon, Frederick T., et al. "The insecurity of wireless networks." *IEEE Security & Privacy* 10.4 (2012): 54-61.

[13] Stimpson, Thomas, et al. "Assessment of security and vulnerability of home wireless networks." *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. IEEE, 2012.

[14] Li, Jiang, and Moses Garuba. "Encryption as an effective tool in reducing wireless LAN vulnerabilities." *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*. IEEE, 2008.

[15] Thanthrige, Udaya Sampath K. Perera Miriya, Jagath Samarabandu, and Xianbin Wang. "Machine learning techniques for intrusion detection on public dataset." *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on*. IEEE, 2016.

[16] Alotaibi, Bandar, and Khaled Elleithy. "A majority voting technique for wireless intrusion detection systems." *Long Island Systems, Applications and Technology Conference (LISAT), 2016 IEEE*. IEEE, 2016.

[17] Aminanto, Muhamad Erza, and Kwangjo Kim. "Detecting Impersonation Attack in WiFi Networks Using Deep Learning Approach." *International Workshop on Information Security Applications*. Springer, Cham, 2016.

[18] Kaur, Navpreet, and Sangeeta Monga. "Comparisons of wired and wireless networks: A review." *International Journal of Advanced Engineering Technology* 5.2 (2014): 34-35.

[19] Wi-Fi structure.http://theelitezone.weebly.com/wifi.html(Accessed on 05/04/2017)

[20] Vibhuti, Shivaputrappa. "IEEE 802.11 WEP (Wired Equivalent Privacy) concepts and vulnerability." Accessed on 29 (2008).

[21] KDD Cup 99.http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.(Accessed on 05/02/2017)

[22] Jha, Jayshree, and Leena Ragha. "Intrusion detection system using support vector machine." *International Journal of Applied Information Systems (HAIS)-ISSN* (2013): 2249-0868.

[23] Lakshmi, P., and D. Geetha. "Intrusion Detection System Using Modified Support Vector Machine." *Networking and Communication Engineering* 7.10 (2015): 430-434.

[24] Salama, Mostafa, et al. "Hybrid intelligent intrusion detection scheme." *Soft computing in industrial applications* (2011): 293-303.

[25] Revathi, S., and A. Malathi. "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection." (2013).

[26] AWID-Attributes.http://icsdweb.aegean.gr/awid/attributes.html. (Accessed on 05/02/2017)

[27] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[28] Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." *Advances in neural information processing systems*. 2007.

[29] Hinton, Geoffrey. "A practical guide to training restricted Boltzmann machines." *Momentum* 9.1 (2010): 926.

[30] Fischer, Asja, and Christian Igel. "An introduction to restricted Boltzmann machines." *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* (2012): 14-36.

[31] Larochelle, Hugo, and Yoshua Bengio. "Classification using discriminative restricted Boltzmann machines." *Proceedings of the 25th international conference on Machine learning*. ACM, 2008.

[32] Schalkoff, Robert J. Artificial neural networks. Vol. 1. New York: McGraw-Hill, 1997.

[33] Rojas, Raul. "The backpropagation algorithm." Neural networks. Springer Berlin Heidelberg, 1996. 149-182.

[34] Yam, Y. F., and T. W. S. Chow. "Extended backpropagation algorithm." Electronics Letters 29.19 (1993): 1701-1702.

[35] Refaeilzadeh, Payam, Lei Tang, and Huan Liu. "Cross-validation." *Encyclopedia of database systems*. Springer US, 2009. 532-538.

[36] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." *Ijcai*. Vol. 14. No. 2. 1995.

[37] Marlin, Benjamin, et al. "Inductive principles for restricted Boltzmann machine learning." *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.

[38] Ct, Marc-Alexandre, and Hugo Larochelle. "An infinite restricted Boltzmann machine." *Neural computation* (2016).

[39] Wang, Shuo, and Xin Yao. "Relationships between diversity of classification ensembles and single-class performance measures." *IEEE Transactions on Knowledge and Data Engineering* 25.1 (2013): 206-219.

[40] Sokolova, Marina, and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." *Information Processing & Management* 45.4 (2009): 427-437.

[41] Phung, Son Lam, Abdesselam Bouzerdoum, and Giang Hoang Nguyen. "Learning pattern classification tasks with imbalanced data sets." (2009): 193.

[42] Ferri, Csar, Jos Hernndez-Orallo, and R. Modroiu. "An experimental comparison of performance measures for classification." *Pattern Recognition Letters* 30.1 (2009): 27-38.

[43] Md Sohail Ahmad and Vivek Ramachandran. *Cafe latte with a free topping of cracked wep retrieving wep keys from road warriors*. 2007.

[44] Hirte Attack. Nov. 2014. URL: http://www.aircrack-ng.org/doku.php?id=airbase-ng#hirte attack in access point mode.

[45] Noor Al-Gharabally, Nosayba El-Sayed, Sara Al-Mulla,and Imtiaz Ahmad. Wireless honeypots: survey and assessment. In: *Proceedings of the 2009 conference on Information Science, Technology and Applications*. ACM. 2009, pp. 4552.

[46] Yimin Song, Chao Yang, and Guofei Gu. Who is peeping at your passwords at Starbucks?To catch an evil twin access point. In: *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*.IEEE. 2010, pp. 323332.

[47] Kemal Bicakci and Bulent Tavli. Denial-of-Service attacks and countermeasures in IEEE 802.11 wireless networks. In: *Computer Standards & Interfaces* 31.5 (2009), pp. 931941.

[48] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In: Information Security Applications. Springer, 2007, pp. 188202.

[49] KoreK. ChopChop (Experimental WEP attacks). Nov. 2014. URL: http://www.netstumbler.org/showthread. php?t=12489.

[50] Andrea Bittau. The Fragmentation Attack in Practice. In: IEEE Symposium on Security and Privacy, IEEE Computer Society. 2005.

Appendix A

System Implementation Codes

A.1 Deep Belief Networks

```
%%  deep belief network
classdef DBN
    properties
        model;
        classifier = false; % rbm of dbn classifier or not
        num_of_layers;   % total number of layer in dbn
        num_of_rbm_layers;
        rbm_layers; %list of rbm layers in dbn
    end
    methods
        function obj = DBN(model)
            obj.model = model;
            obj.num_of_layers = numel(model.size);
            obj.num_of_rbm_layers = obj.num_of_layers-1;
            obj.classifier = model.classifier;
            for il = 2:obj.num_of_layers  %% create dbn with
               its parameters
                layer_model.size = [model.size(il-1),model.
                   size(il)];
```

```
18                  obj.rbm_layers{il-1} = RBM(layer_model);
19                  obj.rbm_layers{il-1}.learning_rate = model.
                        learning_rate(il-1);
20                  obj.rbm_layers{il-1}.num_epoch = model.
                        num_epoch(il-1);
21                  if obj.classifier && (il == obj.num_of_layers
                        )
22                      obj.rbm_layers{end}.classifier = true;
23                  end
24              end
25          end
26          %% train deep belief network
27          function obj = train_dbn(obj, attribute, target)
28              for itL = 1:obj.num_of_rbm_layers
29                  disp(['Training ' num2str(itL) '/' num2str(
                        obj.num_of_rbm_layers) ' RBM of DBN']);
30                  fprintf('\n');
31                  % if it is above 1st rbm initialize the
                        weight from
32                  % the transpose of the previous rbm weight
33                  if itL >=2
34                      obj.rbm_layers{itL}.weight = obj.
                            rbm_layers{itL-1}.weight(1:obj.
                            rbm_layers{itL}.num_hid, 1:obj.
                            rbm_layers{itL}.num_vis)';
35                  end
36                  % if it is classifier train in discriminative
                         else generative way
37                  if itL == obj.num_of_rbm_layers && obj.
                        classifier
38                      obj.rbm_layers{itL} = obj.rbm_layers{itL
                            }.train_rbm(attribute, target);
```

```
39                  else
40                      obj.rbm_layers{itL} = obj.rbm_layers{itL
                            }.train_rbm(attribute, []);
41                  end
42                  % take the output currnt rbm for input of
                        next rbm
43                  attribute = obj.rbm_layers{itL}.hid_expect(
                        attribute);
44              end
45          end
46          %% fine tune dbn network using back propagation
                algorithm
47          function bpnn = fine_tune(obj,data, target, ft_model)
48              %% prepare the model
49              num_outputs = size(oneOfK(target), 2);
50              ft_model.size = [obj.model.size, num_outputs];
51              %% create bpnn object
52              bpnn = back_prop_nueral_net(ft_model);
53              %% transfer dbn parameters to dpnn
54              for itL = 1:numel(obj.model.size) - 1
55                  bpnn.Layers{itL}.weight = obj.rbm_layers{itL
                        }.weight';
56                  bpnn.Layers{itL}.bias = obj.rbm_layers{itL}.
                        hid_bias';
57              end
58              %% train bpnn
59              bpnn = bpnn.train_back_prop(data, oneOfK(target))
                    ;
60          end
61          %% classify using pre-trained DBN
62          function [conf, pred,error,misClass] = classify(obj,
                data,target)
```

```
63              if isvector(target);
64                  target = oneOfK(target);
65              end
66              % propagate the data to each RBM layers
67              for iL = 1:obj.num_of_rbm_layers-1
68                  data = obj.rbm_layers{iL}.hid_expect(data);
69              end
70              % get the last rbm hidden activation
71              num_class = obj.rbm_layers{end}.num_class;
72              tmp = obj.rbm_layers{end}.hid_vis_sampling(data,
                   zeros(size(data,1),num_class));
73              % calculate the class probabiity
74              pClass = tmp.sigmoid(bsxfun(@plus,tmp.hid_act*tmp
                   .class_weight',tmp.class_bias));
75              % classify the computed class probability
76              [~, pred] = max(pClass,[],2);
77              % classify the computed class probability
78              [~,target] = max(target,[],2);
79              %compute the confusion matrix
80              conf = confusionmat(pred,target);
81              %compute classification error
82              misClass = find(pred~=target);
83              error = numel(misClass)/size(data,1);
84          end
85      end
86  end
```

A.2 Restricted Boltzmann Machine

```
%%  define restricted boltzmann machine
classdef RBM
    properties
        classifier = false; % is the current rbm classifier
           not
        num_obs;            % number of training observation
        num_vis;            % number of visible units
        num_hid;             % number of hidden units
        num_class;          % number of class units
        weight;             % network weight
        weight_grad;        % weight gradiant
        class_weight;       % classifier weight
        class_weight_grad;
        weight_decay =0.000001;     %0.000001;
        vis_bias;                   % visible layer bias
        vis_bias_grad;
        hid_bias;                % hidden layer bias
        hid_bias_grad;
        class_bias;              % class bias
        class_bias_grad;
        log;
        vis_act;           %visibla layer activation
        hid_act;           %hidden layer activation
        hid_act_0;
        class_act;          %class activation
        learning_rate = 0.1;         %learning rate
        momentum = 0.5;
        num_epoch = 100;
        batch_indx = [];    %batch indeices in the data
        batch_size = 100;
```

```
30          num_gibbs = 1;
31          vis_sample = 0;
32          hid_sample = 1;
33      end
34      methods
35          %% rbm constractor
36          function obj = RBM(model)
37              %set rbm parameter from the model
38              obj.num_vis = model.size(1);
39              obj.num_hid = model.size(2);
40              if isfield(model,'classifier');
41                  obj.classifier = model.classifier;
42              end
43              obj.weight = 0.01*randn(obj.num_vis, obj.num_hid)
                    ;
44              obj.weight_grad = zeros(size(obj.weight), 'single
                    ');
45              obj.vis_bias = zeros(1,obj.num_vis,'single');
46              obj.vis_bias_grad = zeros(size(obj.vis_bias),'
                    single');
47              obj.hid_bias = zeros(1,obj.num_hid,'single');
48              obj.hid_bias_grad = zeros(size(obj.hid_bias),'
                    single');
49          end
50          %%
51          function obj = train_rbm(obj, data, target)
52              if notDefined('target')
53                  target = 0;
54              end
55              obj.num_obs = size(data, 1);
56              is true or if target is
57              if obj.classifier || any(target)  % initalize
```

```
            classifier
58              obj.classifier = true;
59              [obj,target] = obj.init_classifer(target);
60          end
61          obj.log.err = zeros(1,obj.num_epoch);
62          obj.batch_indx = obj.create_training_batches(data
                ); %create training batch
63          for ep = 1:obj.num_epoch
64              err_sum = 0;
65              for i = 1:numel(obj.batch_indx)  % train the
                    rbm
66                  curr_batch = data(obj.batch_indx{i},:);
67                  if obj.classifier
68                      curr_batch_target = target(obj.
                            batch_indx{i},:);
69                  else
70                      curr_batch_target = 0;
71                  end
72                  %do gibbs sampling for the current
                        training batch
73                  obj = obj.gibbs_sampling(curr_batch,
                        curr_batch_target);
74                  %update rbm properties
75                  obj = obj.update_rbm(curr_batch,
                        curr_batch_target);
76                  %calculate the error
77                  err_sum = obj.mean_squere_error(
                        curr_batch, err_sum);
78              end
79              %log epoch training data error
80              obj.log.err(ep) = err_sum/obj.num_obs;
81              disp(['epoch ' num2str(ep) ': error = '
```

```
                num2str(obj.log.err(ep))]);
82          end
83      end
84      %%
85      function [obj, target] = init_classifer(obj, target)
86          %%
87          if isvector(target)
88              target = oneOfK(target);
89          end
90          obj.num_class = size(target,2);
91          obj.class_weight = single(0.1*randn(obj.num_class
               , obj.num_hid));
92          obj.class_weight_grad =  zeros(size(obj.
               class_weight),'single');
93          obj.class_bias = zeros(1,obj.num_class,'single');
94          obj.class_bias_grad = obj.class_bias;
95      end
96      %% Create minibatches
97      function bth_idx = create_training_batches(obj, data)
98          numobs = size(data,1);
99          num_batches = ceil(numobs/obj.batch_size);
100         tmp = repmat(1:num_batches, 1, obj.batch_size);
101         tmp = tmp(1:numobs);
102         randIdx=randperm(numobs);
103         tmp = tmp(randIdx);
104         for i=1:num_batches
105             bth_idx{i} = find(tmp==i);
106         end
107     end
108     %%
109     function obj = gibbs_sampling(obj, data, target)
110         obj = obj.hid_vis_sampling(data, target);
```

```
111             obj.hid_act_0 = obj.hid_act;
112             for i = 1:obj.num_gibbs
113                 obj = obj.vis_hid_sampling(obj.hid_act);
114                 data = obj.vis_act;
115                 obj = obj.hid_vis_sampling(obj.vis_act,
                        target);
116             end
117         end
118         %%
119         function obj = hid_vis_sampling(obj, data, target)
120             if obj.classifier
121                 obj.hid_act = obj.sigmoid(bsxfun(@plus,data*
                        obj.weight + target*obj.class_weight ,obj.
                        hid_bias));
122             else
123                 obj.hid_act = obj.sigmoid(bsxfun(@plus,data*
                        obj.weight, obj.hid_bias));
124             end
125         end
126         %%
127         function obj = vis_hid_sampling(obj, h_act)
128             obj.vis_act = obj.sigmoid(bsxfun(@plus,h_act*obj.
                    weight',obj.vis_bias));
129             if obj.classifier
130                 obj.class_act = obj.softMax(bsxfun(@plus,
                        h_act*obj.class_weight',obj.class_bias));
131             end
132         end
133         %%
134         function obj = update_rbm(obj, data, target)
135             n_obs = size(data, 1);
136             %compute constructive divergence and update the
```

```
    weight and bias
137 cd = (data'*obj.hid_act_0 - obj.vis_act'*obj.
    hid_act)/n_obs;
138 obj.weight_grad = obj.momentum*obj.weight_grad +
    (1-obj.momentum)*cd;
139 obj.weight = obj.weight + obj.learning_rate*obj.
    weight_grad;
140 % applay weight decay if if it is different from
    zero
141 if obj.weight_decay > 0
142     obj.weight = obj.weight - obj.learning_rate*
        obj.weight_decay*obj.weight;
143 end
144 db = mean(data) - mean(obj.vis_act);
145 dc = mean(obj.hid_act_0) - mean(obj.hid_act);
146 obj.vis_bias_grad = obj.momentum*obj.
    vis_bias_grad + obj.learning_rate*db;
147 obj.vis_bias = obj.vis_bias + obj.vis_bias_grad;
148 obj.hid_bias_grad = obj.momentum*obj.
    hid_bias_grad + obj.learning_rate*dc;
149 obj.hid_bias = obj.hid_bias + obj.hid_bias_grad;
150 if obj.classifier
151     %% update classifier weight and bias if rbm
        is classifier
152     cd_class=(target'*obj.hid_act_0 - obj.
        class_act'*obj.hid_act)/n_obs;
153     obj.class_weight_grad=obj.momentum*obj.
        class_weight_grad+obj.learning_rate*(
        cd_class-obj.weight_decay*obj.class_weight
        );
154     obj.class_weight = obj.class_weight + obj.
        class_weight_grad;
```

```
155                     dd = (sum(target) - sum(obj.class_act))/n_obs
                           ;
156                     obj.class_bias_grad = obj.momentum*obj.
                           class_bias_grad + obj.learning_rate*dd;
157                     obj.class_bias = obj.class_bias + obj.
                           class_bias_grad;
158                 end
159             end
160             %%
161             function err = mean_squere_error(obj, data, err_0)
162                 err = sum(sum((data-obj.vis_act).^2));
163                 err = err + err_0;
164             end
165             %% Calculate hidden unit expectations for network
166             function exp = hid_expect(obj,data)
167                 exp = obj.sigmoid(bsxfun(@plus,data*obj.weight,
                       obj.hid_bias));
168             end
169             %% Siftmax activation function for input X
170             function c = softMax(obj,X)
171                 c = bsxfun(@rdivide,exp(X),sum(exp(X),2));
172             end
173             %% Sigmoid activation function
174             function p = sigmoid(obj,data)
175                 p = 1./(1 + exp(-data));
176             end
177     end
178 end
```

A.3 Back Propagation Neural Network

```
%%  back propagation neural network
classdef back_prop_nueral_net
    properties
        model
        nLayers;    %number of layers
        Layers;     %back prop layer structure
        epoch = 30;     %training epoch
        mse;    %training mean squear error per batch
        training_cost;      %total training cost for training
            data in each epoch
        net_error;          %network error
        lRate = 0.75;
        momentum = 0.4;     %momuntom
        batch_size = 20;     %single training batch size
        batch_indices;      %training batch indices
        weight_lower_bound = -0.1;
        weight_upper_bound = 0.1;
        weight_decay;               %weight decay rate
        begin_weight_decay = Inf;
    end
    %%
    methods
        function obj = back_prop_nueral_net(model)
            obj.model = model;
            obj.nLayers = numel(model.size);
            obj.Layers{1}.type = 'input';
            for i = 2:obj.nLayers
                if i == obj.nLayers
                    obj.Layers{i}.type = 'output';
                else
```

```
30                  obj.Layers{i}.type= 'hidden';
31              end
32              obj.Layers{i-1}.lRate = obj.lRate;
33              obj.Layers{i - 1}.weight = zeros(obj.model.
                    size(i), obj.model.size(i-1));
34              obj.Layers{i - 1}.weight_grad = zeros(size(
                    obj.Layers{i - 1}.weight));
35              obj.Layers{i - 1}.bias = zeros( obj.model.
                    size(i), 1);
36              obj.Layers{i - 1}.bias_grad = zeros( size(
                    obj.Layers{i - 1}.bias));
37          end
38      end
39      %% train the back propagation algorithm here
40      function obj = train_back_prop(obj, data, target)
41          obj.batch_indices = obj.create_batches(data);
42          obj.mse = zeros(obj.epoch,1);
43          for ep = 1:obj.epoch
44              err_sum = 0;
45              for b = 1:numel(obj.batch_indices)  %train
                    for given batch
46                  batch_data = data(obj.batch_indices{b},:)
                        ;
47                  batch_target = target(obj.batch_indices{b
                        },:);
48                  obj = obj.feed_forward(batch_data,
                        batch_target);
49                  obj = obj.back_prop;
50                  err_sum = err_sum + obj.mse;
51              end
52              obj.training_cost(ep) = err_sum/size(data,1);
53              disp(['Epoch ' num2str(ep) '/' num2str(obj.
```

```
                epoch) ': error = ' num2str(obj.
                training_cost(ep))]);
54          end
55      end
56      %% feed forward to propaget the input data
57      function [obj, out] = feed_forward(obj, data, target)
58          obj.Layers{1}.act = data;
59          for i = 2:obj.nLayers   %starting from the second
                layer
60              sum = bsxfun(@plus, obj.Layers{i-1}.act * obj
                    .Layers{i-1}.weight', obj.Layers{i - 1}.
                    bias');
61              obj.Layers{i}.act = obj.sigmoid(sum);
62          end
63          if ~isempty(target)
64              [obj.mse, obj.net_error] = obj.calculate_mse(
                    target);
65          end
66      end
67      %%
68      function obj = back_prop(obj)
69          %calculate the derivative of output of output
                layer
70          act_der = obj.sigmoid_derivative(obj.Layers{end}.
                act);
71          %calcuate and hold the partial derivative of the
                error
72          partial_der{obj.nLayers} = obj.net_error.*act_der
                ;
73          %go through each layer
74          for i = (obj.nLayers - 1):-1:2
75              % layer error contribution
```

```
76                  prop_error = partial_der{i + 1}*obj.Layers{i
                        }.weight;
77                  %calculate the derivative of given layer
                        activation
78                  act_der = obj.sigmoid_derivative(obj.Layers{i
                        }.act);
79                  %calcuate and hold the partial derivative of
                        the error
80                  partial_der{i} = prop_error.*act_der;
81              end
82              for i = 1:(obj.nLayers - 1)
83                  obj.Layers{i}.dW = (partial_der{i + 1}' * obj
                        .Layers{i}.act)/size(partial_der{i + 1},
                        1);
84                  obj.Layers{i}.db = sum(partial_der{i + 1})'/
                        size(partial_der{i + 1}, 1);
85                  weight_decay_tmp = 0;
86                  if obj.weight_decay > 0
87                      % L2-weight decay
88                      weight_decay_tmp = obj.Layers{lL}.weight*
                            obj.weight_decay;
89                  end
90                  obj.Layers{i}.weight_grad = obj.momentum*obj.
                        Layers{i}.weight_grad + obj.Layers{i}.
                        lRate*(obj.Layers{i}.dW + weight_decay_tmp
                        );
91                  obj.Layers{i}.bias_grad = obj.momentum*obj.
                        Layers{i}.bias_grad + obj.Layers{i}.lRate*
                        obj.Layers{i}.db;
92                  % update the weight parameters
93                  obj.Layers{i}.weight = obj.Layers{i}.weight -
                        obj.Layers{i}.weight_grad;
```

```
94                  obj.Layers{i}.bias = obj.Layers{i}.bias - obj
                        .Layers{i}.bias_grad;
95              end
96          end
97          %%
98          function [conf, pred, err] = classification(obj, data
                , target)
99              obj = obj.feed_forward(data, target);
100             pred = obj.Layers{end}.act;
101             err = obj.calculate_mse(target);
102             [~, sss] = max(pred,[],2);
103             [~,tr] = max(target,[],2);
104             conf = confusionmat(sss,tr);
105         end
106         %% calculate the mean squer error and network error
                here
107         function [mse, net_error] = calculate_mse(obj,
                batch_class_data)
108             %get output of the network for the given training
                    batch
109             network_out = obj.Layers{end}.act;
110             %get the output size
111             num_out = size(network_out, 1);
112             delta = batch_class_data - network_out;
113             %calculate the mse
114             mse = 0.5*sum(sum(delta.^2))/num_out;
115             net_error = -delta;
116         end
117         %% create training batch
118         function batchIdx = create_batches(obj,input_data)
119             num_data = size(input_data,1);
120             nBatches = ceil(num_data/obj.batch_size);
```

```
121             tmp = repmat(1:nBatches, 1, obj.batch_size);
122             tmp = tmp(1:num_data);
123             randIdx=randperm(num_data);
124             tmp = tmp(randIdx);
125             for iB=1:nBatches
126                 batchIdx{iB} = find(tmp==iB);
127             end
128         end
129         %% Sigmoid activation function
130         function p = sigmoid(obj,X)
131             p = 1./(1 + exp(-X));
132         end
133         %% Sigmoid activation function derivative
134         function act = sigmoid_derivative(obj, data)
135             act = data.*(1-data);
136         end
137     end
138 end
```