TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

THESIS NO: 072/MSCS/662

A MapReduce-based Deep Belief Network for Intrusion Detection

by

Raju Shrestha

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFULLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

NOVEMBER, 2017

A MAPREDUCE-BASED DEEP BELIEF NETWORK FOR INTRUSION DETECTION

BY:

RAJU SHRESTHA

072/MSCS/662

SUPERVISED BY:

Prof. Dr. SUBARNA SHAKYA

A THESIS SUBMITTED TO DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INSTITUTE OF ENGINEERING, PULCHOWK CAMPUS

TRIBHUVAN UNIVERSITY

LALITPUR, NEPAL

NOVEMBER, 2017

COPYRIGHT

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

APPROVAL PAGE

The undersigned certify that it has been read and recommended to the Department of Electronics and Computer Engineering for acceptance, a report of thesis entitled **"A MapReduce-based Deep Belief Network for Intrusion Detection"**, submitted by **Mr. Raju Shrestha** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"**.

_____

Prof. Dr. Subarna Shakya
Supervisor,
Department of Electronics and Computer Engineering,
Pulchowk Campus,
Institute of Engineering.

_____

Dr. Pradip Paudyal
External Examiner,
Assistant Director,
Nepal Telecommunications Authority.

_____

Prof. Dr. Subarna Shakya
Committee Chairperson
Department of Electronics and Computer Engineering,
Pulchowk Campus,
Institute of Engineering.

Date of Approval: 10th NOVEMBER, 2107

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

Ananda Niketan, Pulchowk, Lalitpur, P.O. Box 1175, Kathmandu, Nepal.

Tel : 5534070, 5521260 extn. 311, Fax : 977-7-5553946, E-mail : ece@ioe.edu.np

Our Ref :

DEPARTMENTAL ACCEPTANCE

The thesis entitled **"A MapReduce-based Deep Belief Network for Intrusion Detection"**, submitted by **Mr. Raju Shrestha** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"** has been accepted as a bonafide record of work independently carried out by him in the department.

_____

Dr. Dibakar Raj Pant
Head of Department,
Department of Electronics and Computer Engineering,
Pulchowk Campus,
Institute of Engineering,
Tribhuvan University,
Pulchowk, Nepal.

ACKNOWLEDGEMENT

First of all, I would like to recognize my deepest gratitude to my supervisor, Prof. Dr. Subarna Shakya, for his kind assistance, guidance and suggestions throughout the duration of my thesis work.

I would like to express my heartfelt gratitude to HOD Dr. Dibakar Raj Pant and MSCSKE Program Coordinator, Dr. Aman Shakya, for providing me with the opportunity to perform this thesis work. I am also thankful to Prof. Dr. Shashidhar Ram Joshi, Dr. Sanjeeb Prasad Panday, Dr. Basanta Joshi and other faculty members of the Department for their suggestions and inspirations.

I would also like to thank my friends and colleagues for their support and encouragement. Last but not the least, I wish to record my appreciation to all the people who directly or indirectly contributed their help during the course of the thesis work.

ABSTRACT

The network security is being a challenging task in this modern era of IT with increasing number of hacking tools, complexity of networks and network security threats. The efficient and reliable intrusion detection system is an essential need in recent growing digital world. Deep Belief Networks (DBNs) with Restricted Boltzmann Machines (RBMs) as the building block have recently attracted wide attention to the field of Network Intrusion Detection System (NIDS) due to their great performance. DBN consists of two phase of training: first is pre-training of stack of RBMs followed by fine-tuning using backpropagation However, the sequential implementation of DBN is computationally very time consuming to process large amount of data sets. So, this thesis research is focused on implementing a MapReduce-based Deep Belief Network (MRDBN) for distributed computation for efficient NIDS using Hadoop ecosystem. The performance of system has been evaluated using two different datasets: UNSW-NB15 and NSL-KDD. First, the dataset is preprocessed to convert all attributes into numerical values and normalized it. Then, the distributed DBN based on MapReduce framework is implemented and evaluated in preprocessed datasets. The scalability test of system showed that training time for 10 nodes cluster sized MRDBN for intrusion detection is 2.2 times faster than 4 nodes cluster sized MRDBN. The overall accuracy of the MRDBN intrusion detection system for multiclass classification is 82.61% with 3.9% false alarm rate for UNSW-NB15 dataset. The system is found to be more precise than existing Artificial Neural Network (ANN) and Support Vector Machine (SVM) in the field of intrusion detection.

Keywords: *Deep Belief Network, Intrusion Detection System, MapReduce, Deep Learning, Restricted Boltzmann Machine.*

TABLE OF CONTENTS

COPYRIGHT iii
APPROVAL PAGE iv
DEPARTMENTAL ACCEPTANCE v
ACKNOWLEDGEMENT vi
ABSTRACT vii
TABLE OF CONTENTS viii
LIST OF FIGURES x
LIST OF TABLES xii
LIST OF ACRONYMS xiv
1. INTRODUCTION 1
1.1. Background 1
1.2. Problem Statements 4
1.3. Objectives 5
1.4. Scope of Thesis 5
1.5. Organization of Thesis Report 5
2. LITERATURE REVIEW 7
3. RELATED THEORY 10
3.1. Restricted Boltzmann Machine (RBM) 10
3.1.1. Training RBM 11
3.2. Deep Belief Network 15
3.3. MapReduce Framework 16
4. RESEARCH METHODOLOGY 19
4.1. Research Work Flow 19
4.1.1. Data Collection 20

4.1.2. Data Preprocessing 25
4.1.3. Distributed System Design 27
4.1.4. Performance Evaluations Metrics 34
4.2. System Operation 35
4.3. Experimental Setup and Tools 36
5. RESULTS, ANALYSIS AND COMPARISONS 40
5.1. Scalability and Reconstruction Error Test of System 40
5.2. Performance on UNSW-NB15 Dataset 41
5.2.1. Evaluation and Analysis 43
5.2.2. Comparison with ANN 44
5.2.3. Comparison with SVM 45
5.2.4. Comparison between MRDBN, SVM and ANN 46
5.2.5. MRDBN Binary Classification on UNSW-NB15 48
5.3. Performance on NSL-KDD Dataset 48
5.3.1. Evaluation and Analysis 49
5.3.2. Comparison between MRDBN, SVM and ANN 50
5.3.3. MRDBN Binary Classification on NSL-KDD 52
5.4. Cloud-Computing Service Execution Results 52
6. CONCLUSIONS 55
7. LIMITATIONS 56
REFERENCES 57

LIST OF FIGURES

Figure 1.1: Common IDS approach 2

Figure 1.2: Restricted Boltzmann Machine (RBM) 3

Figure 1.3: An illustration of DBN stacked with RBM 4

Figure 3.1: Contrastive Divergence 14

Figure 3.2: An overview of MapReduce framework 17

Figure 4.1: Research work flow 19

Figure 4.2: UNSW-NB15 input dataset after preprocessing 26

Figure 4.3: Intrusion detection system intermediate operations 35

Figure 4.4: Instances (nodes) running in AWS cloud computing platform 38

Figure 4.5: Amazon Elastic MapReduce cluster information dashboard 39

Figure 5.1: Training time versus sample for various cluster size 40

Figure 5.2: Reconstruction error vs. epoch 41

Figure 5.3: Performance comparison between MRDBN and ANN for UNSW-NB15 45

Figure 5.4: Performance comparison between MRDBN and SVM for UNSW-NB15 46

Figure 5.5: Comparative Analysis of MRDBN, SVM and ANN for UNSW-NB15 ... 47

Figure 5.6: Accuracy comparison by attack class of MRDBN, SVM and ANN for UNSW-NB15 47

Figure 5.7: MRDBN prediction output performance for binary classification of UNSW-NB15 48

Figure 5.8: Comparative Analysis of MRDBN, SVM and ANN for NSL-KDD 51

Figure 5.9: Accuracy comparison by attack class of MRDBN, SVM and ANN for NSL-KDD 51

Figure 5.10: MRDBN prediction output performance for binary classification of NSL-KDD 52

Figure 5.11: Adding step for running MapReduce job 53

Figure 5.12: Details of Nodes running in cluster 53

Figure 5.13: Summary of applications running in Hadoop ecosystem 54

Figure 5.14: Syslog information of MapReduce job running 54

LIST OF TABLES

Table 4.1: Categorization of attacks 20

Table 4.2: UNSW-NB15 dataset feature details 21

Table 4.3: A part of the UNSW-NB15 dataset distribution 23

Table 4.4: NSL-KDD features and its type 24

Table 4.5: NSL-KDD dataset distribution 25

Table 4.6: Numeric encoding of attribute state 26

Table 5.1: Hyper parameters and architecture for 3 different MRDBN experiments . 41

Table 5.2: Confusion matrix for experiment 1 42

Table 5.3: Confusion matrix for experiment 2 42

Table 5.4: Confusion matrix for experiment 3 42

Table 5.5: Prediction output summary of Experiment 1, 2 and 3 43

Table 5.6: Detail evaluation by attack classes of Experiment 2 43

Table 5.7: Confusion matrix of ANN output prediction for UNSW-NB15 44

Table 5.8: ANN prediction performance on UNSW-NB15 44

Table 5.9: Confusion matrix for SVM output prediction for UNSW-NB15 45

Table 5.10: SVM prediction performance on UNSW-NB15 46

Table 5.11: Confusion matrix of MRDBN binary classification for UNSW-NB15 48

Table 5.12: Confusion matrix of MRDBN output prediction for NSL-KDD 49

Table 5.13: Detail evaluation by attack classes of MRDBN for NSL-KDD 49

Table 5.14: MRDBN prediction performance on NSL-KDD 49

Table 5.15: Confusion matrix of ANN output prediction for NSL-KDD 50

Table 5.16: Confusion matrix of SVM output prediction for NSL-KDD 50

Table 5.17: ANN and SVM prediction performance on NSL-KDD 50

Table 5.18: Confusion matrix of MRDBN binary classification for NSL-KDD..... 52

LIST OF ACRONYMS

ACCS	Australian Centre for Cyber Security
ANN	Artificial Neural Network
API	Application Program Interface
BM	Boltzmann Machine
BSD	Berkeley Software Distribution
CNN	Convolutional Neural Network
DBN	Deep Belief Network
DoS	Denial of Service
EC2	Elastic Cloud Computing
EMR	Elastic MapReduce
FANN	Forward Additive Neural Network
GA	Genetic Algorithm
GPU	Graphical Processing Unit
ID	Intrusion Detection
IDS	Intrusion Detection System
IT	Information technology
MCMC	Markov Chain Monte Carlo
MRDBN	MapReduce-based Deep Belief Network
MRF	Markov Random Field

NIDS	Network Intrusion Detection System
R2L	Remote to Local
RBM	Restricted Boltzmann Machine
S3	Simple Storage Service
SVM	Support Vector Machine
U2R	User to Root

1. INTRODUCTION

1.1. Background

With the recent convergence of information technology (IT), numerous information devices are becoming tremendously complicated. Connected to each other, they continue to create and save important digital data, ushering in an era of big data. However, the likelihood is very high that they may expose valuable information as they transmit much of it through constant communication with each other. A system becomes more vulnerable as more digital devices are connected. Hackers may also target it to steal data, personal information, and industrial secrets and leak them for illegal gains. Though efforts have been made to secure important information, systems that need protection are becoming increasingly complicated, and attack techniques for penetrating a system continue to evolve and develop accordingly. This presents challenges to people and companies. Given these circumstances, attack detection techniques should also be more intelligent and effective than before to combat attacks from hackers, which are also persistently evolving. The existing technologies for detecting abnormal behavior (threats) generally employ detection methods based on event analysis with preset rules. However, they also face limitations such as a lack of information on actual attacks and financial damages incurred by false detection of attacks in the field of security as well as ever-changing environments. This is why they are used only to automatically collect and analyze diverse security event information to assess the risks.

Intrusion Detection (ID) has become the key technology of network security. The concept of ID was proposed by James P. Anderson in 1980 [1]. The goal of Intrusion Detection Systems (IDS) is to identify unusual access or attacks on internal network security [2]. IDSs are security tools used to detect anomalous or malicious activities from inside and outside intruders. Such activities that violate the security policies of the system are considered anomalous and an alert should be raised by the IDS. An intrusion can be an attack from the Internet, attempts from authorized users of the system to gain more privileges, or an authorized user who misuse their privileges. The intrusion detection systems can be classified into three categories: host based, network

based and vulnerability assessment based. A host based IDS evaluates information found on a single or multiple host systems, including contents of operating systems, system files and application files. While network based IDS evaluates information captured from network communications, analyzing the stream of packets traveling across the network. Packets are captured through a set of sensors. Vulnerability assessment based IDS detects vulnerabilities on internal networks and firewall [2]. Hence, an IDS has three basic functions [3]: monitoring information sources, analysis and response. Two typical methods are commonly used in IDS such as clustering and classification. It is difficult and costly to obtain bulk of labeled network connection records for supervised training in the first stage. The clustering analysis has emerged as an anomaly intrusion detection approach in recent years [4]. Clustering is an unsupervised data exploratory technique that partitions a set of unlabeled data patterns into groups or clusters such that patterns within a cluster are similar to each other but dissimilar to other clusters' pattern [4]. Meanwhile, classification is a supervised method to distinguish benign and malicious traffics based on provided data which usually comes from clustering result as shown in Figure 1.1.

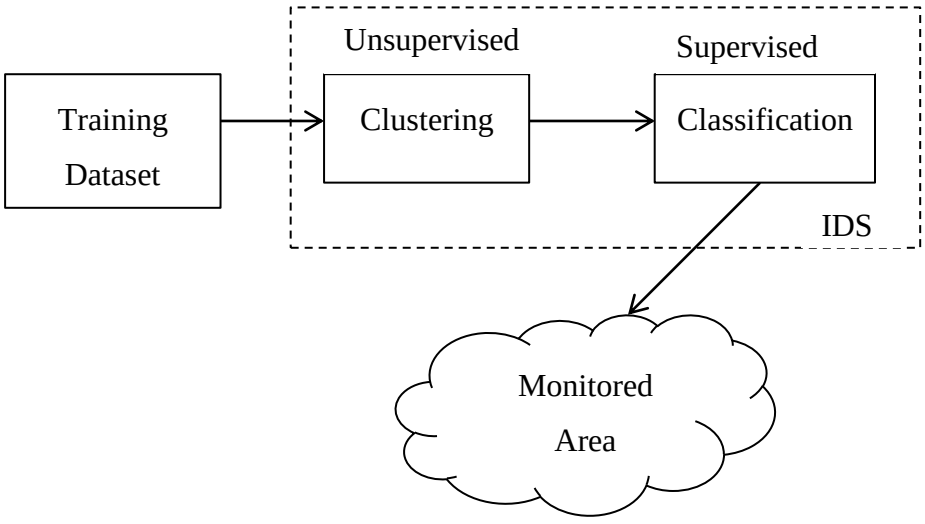

Figure 1.1: Common IDS approach

(Source: Adapted from [5])

Deep learning is one of the most cutting-edge machine learning techniques. It is an important step forward to high-level intelligence, and has shown extremely good performance in many applications in pattern recognition such as image, speech and document processing. Deep models and their learning algorithms are inspired by the

structure and information processing mechanism of human brain. Each deep model has a deep structure that consists of a number of non-linear hidden feature layers and hierarchical feature abstraction mechanism. Restricted Boltzmann Machine (RBM) [6], shown in Figure 1.2, is the building block of the currently developed deep networks, as most of deep networks, such as, Deep Belief network (DBN) [7] shown in Figure 1.3, deep Boltzmann machine [8] and deep neural network discussed in [9], are built with a number of RBMs. A RBM [10] is a two-layer probabilistic graph model, which is constructed with a number of visible and hidden nodes (random variables). Each node has a bias and a connection weight with the nodes in the different layer. The MapReduce framework is based on Hadoop and has become popular in recent years. MapReduce is a type of parallel computing model oriented toward distributed environments. This model provides developers with a complete programming interface, does not require them to understand the computer architecture, and has gradually become a research hotspot for current studies on parallel algorithm design [11].

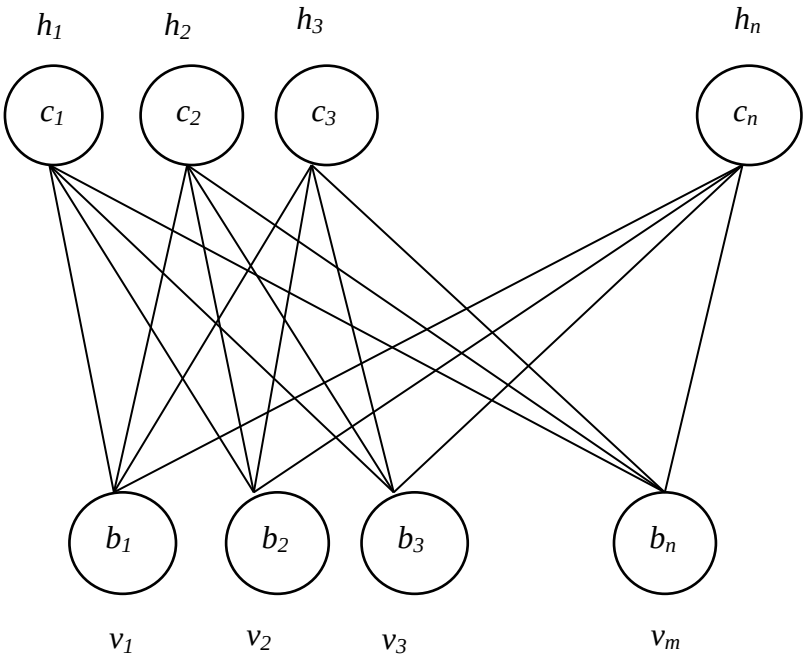

Figure 1.2: Restricted Boltzmann Machine (RBM)

(Source: Adapted from [6])

In this thesis, the latest achievement of neural network algorithm, deep learning [7], is implemented to accelerate the performance of DBN neural network for the intrusion detection based on MapReduce framework. This thesis is mainly focused on combining

deep learning algorithm with cloud computing platform to deal with large-scale network traffic data. A MapReduce-based DBN (MRDBN) is implemented in this thesis to verify the efficiency improvement this mechanism achieved on new practical large-scale UNSW-NB15 [12] dataset for anomaly Network Intrusion Detection System (NIDS). A many-layered neural network could be effectively pre-trained one layer at a time, treating each layer in turn as un unsupervised Restricted Boltzmann Machine, then followed by supervised back-propagation fine-tuning.

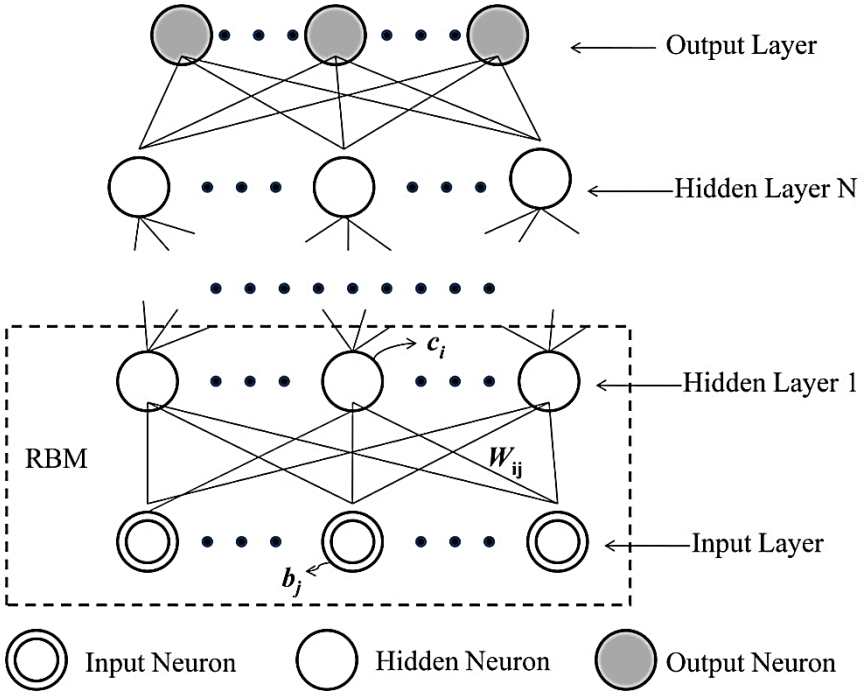

Figure 1.3: An illustration of DBN stacked with RBM

1.2. Problem Statements

Owing to the limited samples and computing cell, expressive power of shallow learning methods for complex function is limited. The primary challenge of shallow neural network is proper feature selections for the network traffic dataset for anomaly detection are difficult. As attack scenarios are continuously changing and evolving, the features selected for one class of attach may not work well for other classes of attacks. Moreover, the deep learning methods are complex and require a lot of time to train properly. The continuous collection of traffic data by the network has leaded to a Big Data problem in network intrusion detection and prediction, therefore, how to develop

an efficient Big-Data-oriented intrusion detection model is a theoretic and practical problem that should be solved.

1.3. Objectives

The main objectives of carrying out this thesis research are:

1. To implement an Intrusion Detection System using MapReduce based Deep Belief Network for large network traffic data with low false alarm rate.
2. To test the system in UNSW-NB15 and NSL-KDD dataset and compare the performance achieved with respect to existing ANN and SVM.

1.4. Scope of Thesis

The scopes of this thesis are as follows:

a) The anomaly based intrusion detection technique is used in this thesis work.
b) It is a feature based prediction system applying MapReduce based distributed Deep Belief Network.
c) A UNSW-NB15 and NSL-KDD datasets are used for training and testing.
d) Scalability and performance measurements for the MRDBN and comparative study with exiting approaches: ANN and SVM are done in this work.

1.5. Organization of Thesis Report

This report consists of 7 chapters and is organized as follows**:** Chapter 1 is an introductory chapter which includes background of the study, problem statement, objectives of thesis and scope and limitations of the work. Chapter 2 highlights the major works carried out in this same kind of research as literature review. Chapter 3 presents the general overview of the Deep Learning, its Concept and Components. It describes about Restricted Boltzmann Machine (RBM) and Contrastive Divergence for its training strategy. Also, The Deep Belief Network is described by constructing a

stack of RBM. Chapter 4 presents the methodology of implementing DBN in MapReduce based framework. Moreover, this chapter describes the research flow and methodology followed for developing intrusion detection system and its experimental setup. The result evaluations, analysis and comparisons are discussed in chapter 5. Chapter 6 is incorporates the conclusion and future extension of the thesis work. Chapter 7 is the last chapter that describes the limitations of the work.

2. LITERATURE REVIEW

Commonly, intrusion is an act of encroaching or infringing the reliability, confidentiality or avoiding the accessibility of a resource [13]. Through internet, Intrusions Detection Systems discovers illegal or malicious assaults over a computer system which happens mainly. By the safety and hope of a system, these assaults can be compromised. To perceptive files, these harasses can acquire quite a few forms like network attack against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, illegal logins and access. IDSs can be categorized as misuse detectors or anomaly detectors by sorting out broadly based on their models of detection.

Mehdi M. and Mohammed Z. [14] were the first to explore detection systems which detect not only attacks but also attack types. They used networks made of two or three layers. One the shortcomings with their model are overlearning, that is the inability of the network to detect new attacks. It is a problem that has to do with the generalization of the network. They notice that detection with a three-layer network is more reliable than a two-layer one. The main weakness here is the number of attack types that their model can detect. Certain authors lifted that limitation and went further beyond.

In the very next year, using the K-means clustering algorithm a technique have built up by K. M. Faraoun and A. Boukelif [15] to improve the learning capacities and decrease the computation strength of a competitive learning multi-layered neural network. Through a back propagation learning means the recommended model used multi-layered network structural design. To decrease the amount of examples to be offered to the neural network, the K-means algorithm was initially used to the training dataset by automatically choosing a most favorable set of samples. The acquired results showed that the suggested technique executes specially in terms of both precision and computation time when pertained to the KDD99 dataset match up to a normal learning schema that utilized the full dataset.

In advanced to the above approach, Iftikhar Ahmad,Azween et al. have proposed a paper to surmount presentation issues an optimized interference detection mechanism by means of soft computing techniques [16]. The KDD-cup dataset was applied that

was a benchmark for assessing the safety identification mechanisms. To change the key in models into a feature space the Principal Component Analysis (PCA) was applied. Selecting of a suitable quantity of principal components was an important problem. As an alternative of using conventional method, Genetic Algorithm (GA) was applied in the optimum choice of principal components accordingly. The Support Vector Machine (SVM) was employed for categorization reason. In addition, a proportional study was prepared with presented approaches. Therefore, the technique presented optimal interference detection mechanism was proficient to minimize amount of features and maximize the identification rates.

Gaikwad et. al. [17] introduced a technique based on fuzzy clustering and ANN approach. This method could be applicable to overcome the issues of weak stability detection as well as low precision detection. The restore point in this method was employed for registry keys, system files roll back, thesis database and installed programs. Fuzzy clustering will generate different subsets for training in order to reduce the amount of subset size and complexity. Then each subset is trained with different type of artificial neural network and finally processed to obtain significant results. Jaiganesh et. al. [18] suggested a novel back propagation model for intrusion detection. This method makes training pair with a combination of input and equivalent target were generated and implemented into the network. Performance success can be measured by false alarm and detection rate. Detection rate was proven to be less than 80% for U2R, R2L, DoS and Probe attacks. However, the major issue of the method was found to be much inefficient to detect hidden attackers present in the system. MLP method was considered as a failure model due to irrelevant output.

More recently, however, studies are being conducted on intrusion detection with deep learning, an artificial neural network (ANN) algorithm that is an advance from the traditional machine learning in which pattern extraction and learning are separate tasks. Unlike the widely used existing intrusion detection method that produces a rule or model for malicious attack patterns, it finds relationships directly from secured data to detect abnormal risks. N. Gao et. al. studied intrusion detection using deep belief networks (DBNs); a test using the KDD Cup 99 data showed that accuracy increased 6% or more over that of the existing SVM model and ANN model [19]. Another study compared the forward additive neural network (FANN) and SVM. FANN is an algorithm that makes up for the weaknesses of the existing back-propagation algorithm.

Jo carried out an intrusion detection study using FANN and compared the result with the existing SVM model; FANN demonstrated higher accuracy and a better detection rate than the SVM [20]. In addition, Deep Belief Network (DBN) and SVM have used for classification for intuition detection on NSL-KDD dataset where DBN is used for feature extraction and achieved highest accuracy about 92.84% [21].

M. Z. Alom et. al. [22], explored the capabilities of DBN's performing intrusion detection through series of experiments after training it with NSL-KDD dataset. The trained DBN network identifies any kind of unknown attack in dataset supplied to it and is claimed to be the first comprehensive paper performing intrusion detection using deep belief nets. The proposed system not only detect attacks but also classify them in five groups with the accuracy of identifying and classifying network activity based on limited, incomplete, and nonlinear data sources. The proposed system achieved detection accuracy about 97.5% for training 40% of NSL-KDD dataset for only fifty iterations. However, the research still does not meet comprehensive representation of a modern low foot print attack environment, since; the research analysis was carried on NSL-KDD dataset. S. Son et. al. [23] address a novel method to efficiently manage and analyze a large amount of log data using Hadoop ecosystem. They implemented simple 3-sigma techniques to detect anomalies in log data. However, the basic method has a problem of reporting unnecessary and duplicated anomalies in the case where there are rapid changes.

So, to meet the continuously changing and evolving attacks, Big Data of network traffic and limitation of existing approaches, this thesis is carried out. In this thesis work, features selected from network traffic data are preprocessed with normalization and numerical encoding and finally a MapReduce based DBN (MRDBN) is used to classify network intrusion. Also the comparative study with existing approaches: ANN and SVM, is carried out.

3. RELATED THEORY

3.1. Restricted Boltzmann Machine (RBM)

A Boltzmann Machine (BM) is a generative stochastic neural network that can learn a probability distribution over its inputs. A Restricted Boltzmann Machine (RBM) is further restricted to abandon visible-visible and hidden–hidden connections. A RBM is a two layer probabilistic bipartite undirected graph model as illustrated in Figure 1.2, which is constructed with a number of visible and hidden nodes (random variables). Each node has a bias and a connection weight with the nodes in the different layer.

As shown in Figure 1.2, RBM consists of m visible units v = (v_1,v_2, . . . , v_m) representing the observable data, and n hidden units h = (h_1,h_2, . . ,h_n) to capture the dependencies between the observed variables. In binary RBMs, focus in this thesis, the random variables (v, h) takes values (v, h) ∈. $\{0, 1\}^{m+n}$. An RBM is an energy-based probabilistic model, in which the Gibbs probability distribution is defined through an energy function.

Its probability is defined as

$$P(v,h) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z} \tag{3.1}$$

Where the energy function is given by

$$E(v,h) = \sum_i \sum_j w_{i,j} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i \tag{3.2}$$

And Z is partition function which is given by summing over all possible pair of visible and hidden vectors:

$$Z = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \tag{3.3}$$

In Equation 3.2, for all i ∈ {1, . . . , n} and j ∈ {1, . . . , m}, $w_{i,j}$ is a real valued weight associated with the edge between the units v_j and h_i, and b_j and c_i are real valued bias terms associated with the j^{th} visible and the i^{th} hidden variable, respectively.

The graph of an RBM has connections only between the layer of hidden and the layer of visible variables, but not between two variables of the same layer. In terms of probability, this means that the hidden variables are independent given the state of the visible variables and vice versa:

$$P(h_i = 1|v) = sigm\,(\sum_{j=1}^{m} W_{i,j}v_j + c_i) = \frac{1}{1 + e^{(-b_i - W_i v)}} \tag{3.4}$$

And similarly,

$$P(v_j = 1|h) = sigm\,(\sum_{i=1}^{n} W_{i,j}h_i + b_j) \tag{3.5}$$

Thus, due to the absence of connections between hidden variables, the conditional distributions p(h | v) and p(v | h) factorize nicely. The conditional independence between the variables in the same layer makes Gibbs sampling especially easy: instead of sampling new values for all variables subsequently, the states of all variables in one layer can be sampled jointly. Thus, Gibbs sampling can be performed in just two steps: sampling a new state h for the hidden neurons based on p(h | v) and sampling a state v for the visible layer based on p(v | h). This is also referred to as block Gibbs sampling [6].

3.1.1. Training RBM

Unsupervised learning means learning an unknown distribution based on sample data. This includes finding new representations of data that foster learning, generalization, and communication [6]. If it is assumed that the structure of the graphical model is known and that the energy function belongs to a known family of functions parameterized by $\Theta=\{b_j, c_i, w_{ij}\}$ used in Equation 3.2 , unsupervised learning of a data

distribution with an Markov Random Field (MRF) means adjusting the parameters Θ. It is considered the training data S={x_1,…..x_L}. The data samples are assumed to be independent and identically distributed. That is, they are drawn independently from some unknown distribution. The Gibbs distribution of an MRF describes the joint probability distribution of (V,H) and one is usually interested in the marginal distribution of V, which is given by

$$p(v) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(v,h)} \tag{3.6}$$

A standard way of estimating the parameters of a statistical model is maximum-likelihood estimation. Applied to MRFs, this corresponds to finding the MRF parameters that maximizes the maximum the probability of S under the MRF distribution, training corresponds to finding parameter Θ that maximizes the likelihood given training data.

Restricted Boltzmann machines are MRFs with hidden variables and RBM learning algorithms as based on gradient ascent on the log-likelihood. For a model of Equation 3.6 with parameter Θ, the log-likelihood given a single training example v is

$$\ln p(v|\Theta) = \ln \frac{1}{Z} e^{-E(v,h)} = \ln \sum_{\boldsymbol{h}} e^{-E(v,h)} - \ln \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(v,h)} \tag{3.7}$$

And for the gradient,

$$\frac{\partial(\ln p(v|\Theta))}{\partial\Theta} = -\sum_{h} p(h|v)\frac{\partial E(v,h)}{\partial\Theta} + \sum_{v,h} p(v,h)\frac{\partial E(v,h)}{\partial\Theta} \tag{3.8}$$

For RBMs the first term (positive phase) of Equation 3.8 (i.e. the expectation of the energy gradient under the conditional distribution of the hidden variables given a training example **v**) can be computed efficiently because it factorizes nicely.

Using the factorization trick in Equation 3.8, the derivative of the log-likelihood of a single pattern v with respect to the weight $w_{i,j}$ becomes

$$\frac{\partial(\ln p(v|\Theta))}{\partial w_{i,j}} = v_j P(H_i = 1|v) - \sum_{v} p(v)\, v_j P(H_i = 1|v) \tag{3.9}$$

Analogously to Equation 3.9, we get the derivative with respect to the bias parameter b_j of j^{th} visible variable,

$$\frac{\partial(\ln p(v|\Theta))}{\partial b_j} = v_j - \sum_v p(v)\, v_j \tag{3.10}$$

and with respect to the bias parameter c_i of the i^{th} hidden variable.

$$\frac{\partial(\ln p(v|\Theta))}{\partial c_i} = P(H_i = 1|v) - \sum_v p(v)\, P(H_i = 1|v) \tag{3.11}$$

In above Equations 3.9, 3.10 and 3.111, there is the difference between two expectations: the expected values of the energy function under the model distribution and under the conditional distribution of hidden variables given the training example. For calculating the second term (negative phase) of each mentioned Equations is difficult. Directly calculating this sums, which run over all values of the variables, leads to a computational complexity which is in general exponential in number of variables of the MRF. To avoid this computational burden, the expectations can be approximated by samples drawn from the corresponding distributions based on Markov Chain Monte Carlo (MCMC) techniques. This method has a low converging speed, and it is difficult to define an adequate step size during training stage [24]. Also, obtaining unbiased estimates of the log-likelihood gradient using MCMC methods typically requires many sampling steps. These resulted in a long training time for RBM to reach steady state. Hinton et al. suggested using the Contrastive Divergence (CD-k) theory, which shortens the calculation time required while maintaining the same level of accuracy. These days, CD has become a standard way to train RBMs.

The idea of K-step contrastive divergence learning (CD-k) is quite simple instead of approximating the second term in the term in the likelihood gradient by a sample from the RBM-distribution (which would require running a Markov chain until the stationary distribution is reached), a Gibbs chain is run for only k steps .A each step in the Markov chain, visible units are samples given hidden units, hidden units are sampled

given visible units. According to Hinton, The CD learning uses two tricks to speed up the sampling process. The first on it to initialize the Markov chain with a training example, and the second on is to obtain samples, after only k-steps of Gibbs sampling. A lot of experiments show that the performances of the approximations are still very good when k=1.

It is also seen that CD learning provides an approximation of log-likelihood gradient that has been found to be a successful update rule for training probabilistic models. Variation justification can provide a theoretical proof to the convergence of the learning process. Conducting CD-1 learning by using, it is easy to get updating rules for parameter (w_{ij}, b_j, c_i). The pseudo-code is demonstrated in Algorithm 1.

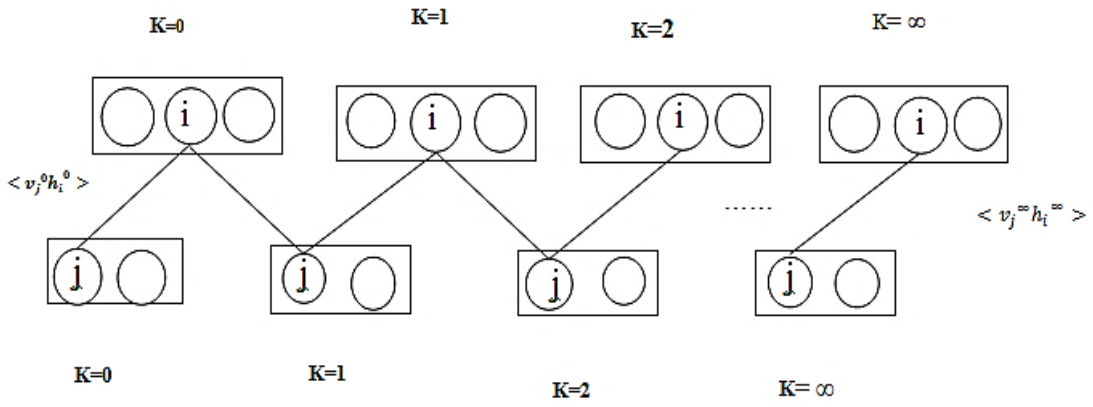

Figure 3.1: Contrastive Divergence

(Source: http://www.cs.toronto.edu/~hinton/csc2535/notes/lec4new.pdf)

Algorithm 1: Updating rules for RBM

Input: $v^{(0)}$ is a training example form training distribution for RBM;

ϵ is the learning rate for updating the parameters.

$W_{i,j}$ is the visible-hidden connection weight matrix.

b_j is the bias vector for input (visible) units.

C_i is the bias vector for hidden units.

Output: The updated parameters in the RBM: $W_{i,j}$, b_j and C_i

for all hidden units i **do**

Compute $p(h_i^{(0)}=1|v^{(0)})$ using Equation 3.4 (positive phase)

Sample $h_i^{(0)} \epsilon (0,1)$ from $p(h_i^{(0)}=1|v^{(0)})$.

end for

for all visible units j **do**

Compute $p(v_j^{(1)}=1|h^{(0)})$ using Equation 3.5 (negative phase)

Sample $v_j^{(1)} \epsilon (0,1)$ from $p(v_j^{(0)}=1|h^{(0)})$.

end for

for all hidden units i **do**

Compute $p(h_i^{(1)}=1|v^{(1)})$ using Equation 3.4

Sample $h_i^{(1)} \epsilon (0,1)$ from $p(h_i^{(1)}=1|v^{(1)})$.

end for

Update:

$W_{ij} \leftarrow W_{ij} + \epsilon(v^{(0)} * p(h_i^{(0)}=1|v^{(0)}) - v^{(1)} * p(h_i^{(1)}=1|v^{(1)}))$

$b_j \leftarrow b_j + \epsilon(v^{(0)} - v^{(1)}))$

$c_i \leftarrow c_i + \epsilon(p(h_i^{(0)}=1|v^{(0)}) - p(h_i^{(1)}=1|v^{(1)})$

return $W_{i,j}$, b_j and C_i

3.2. Deep Belief Network

The DBN is a direct acyclic graph except from the top two layers that form an undirected bipartite graph. The top two layers is what gives the DBN the ability to unroll into a deep autoencoder and perform reconstructions of the input data [25]. The DBN consists of a visible layer, output layer and a number of hidden layers as shown in

Figure 1.3. A DBN is a probabilistic generative model that contains many layers of hidden units. The top two layers form an undirected bipartite graph with the lower layers forming a directed sigmoid belief network as already defined. A DBN can be constructed from stack of RBMs.

The training process of the DBN is defined by two steps: pre-training and fine-tuning. In pre-training the layers of the DBN are separated pair wise to form Restricted Boltzmann Machines (RBM). Each RBM is trained independently, such that the output of the lower RBM is provided as input to the next higher-level RBM and so forth. This way the layers of the DBN are trained as partly independent systems. The goal of the pre-training process is to achieve approximations of the model parameters (that is connection weight between visible and hidden unit, visible unit bias and hidden unit bias). [9] Showed that RBMs can be stacked and trained in a greedy manner to form so-called Deep Belief Networks (DBN). DBNs are graphical models which learn to extract a deep hierarchical representation of the training data. They model the joint distribution between observed vector *v* and the ℓ hidden layers h^k as follows:

$$P(v, h^1, \dots, h^\ell) = \left(\prod_{k=0}^{\ell-2} p(h^k | h^{k+1}) \right) p(h^{\ell-1}, h^\ell) \qquad (3.12)$$

Where, $v=h^{(0)}$, $p(h^{(k-1)}|h^{(k)})$ is a conditional distribution for the visible units conditioned on the hidden units of the RBM at level k which can be easily calculated by using Equation 3.4 and 3.5 , and $p(h^{\ell-1}, h^\ell)$ is the visible-hidden joint distribution in the top-level RBM. This is illustrated in Figure 1.3.

3.3. MapReduce Framework

MapReduce provides a programming paradigm for performing distributed computation on computer clusters [26]. Figure 3.2 gives an over view of the MapReduce framework. In a MapReduce system such as Hadoop, the user program forks a Master controller process and a series of Map tasks (Mappers) and Reduce tasks (Reducers) at different computers (nodes of a cluster). The responsibilities of the Master involve creating some

number of Mappers and Reducers and keeping track of the status of each Mapper and Reducer (executing, complete or idle).

The computation in one MapReduce job consists of two phases, i.e., a map phase and a reduce phase. In the Map phase, the input dataset, stored in a distributed file system (HDFS), is divided into a number of disjoint subsets which are assigned to mappers in terms of <key, value> pairs. In parallel, each Mapper applies the user specified map function to each input <key, value> pair and outputs a set of intermediate <key, value> pairs which are written to local disks of the map computers. The underlying system passes the locations of these intermediate pairs to the master who is responsible to notify the reducers about these locations. In the Reduce phase, when the reducers have remotely read all intermediate pairs, they sort and group them by the intermediate keys. Each Reducer literately invokes a user specified reduce function to process all the values for each unique key and generate a new value for each key. The resulting <key, value> pairs from all of the Reducers are collected as final results which are then written to an output file.

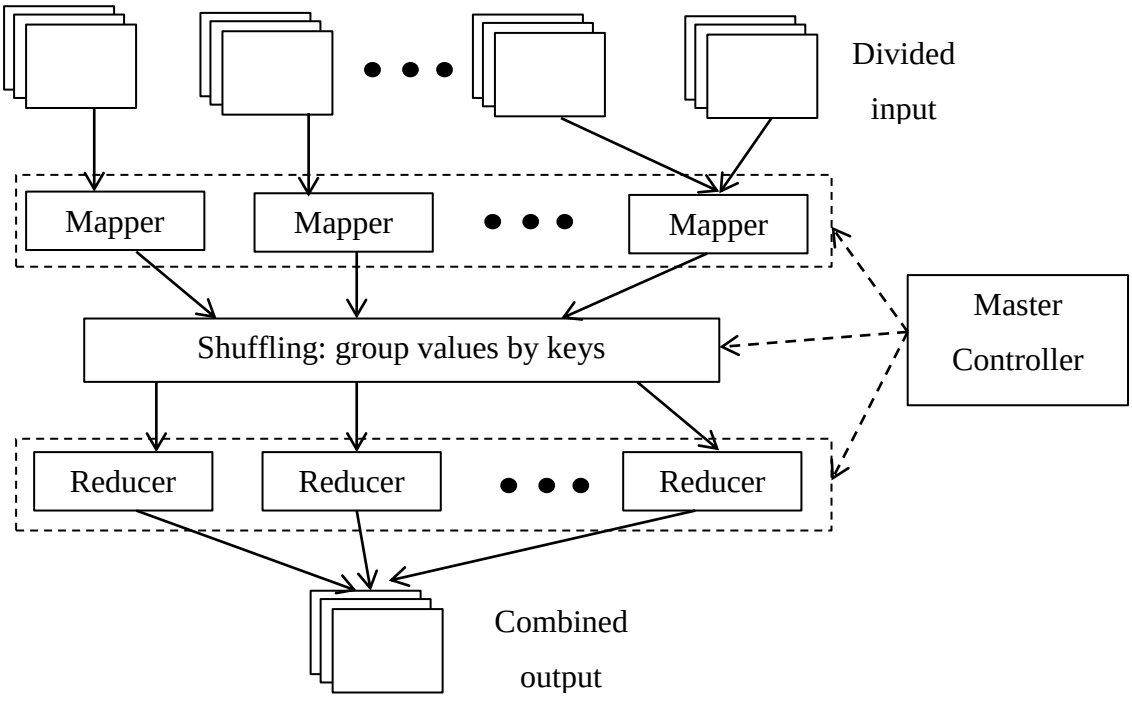

Figure 3.2: An overview of MapReduce framework

(Source: Adapted from [26])

Hadoop is an open-source framework for distributed computing, written in Java and developed by the Apache Foundation and inspired by Google's MapReduce [27]. A

typical Hadoop cluster consists of a master node and any number of computing nodes (data nodes). The purpose of the master is to interact with users, monitor the status of the computing nodes, keep track of load balancing and handle various other background tasks. The computing nodes deal with processing and storing the data. The execution of a MapReduce program (alternatively, a MapReduce job) can briefly be summed up in the following steps:

1. The user uploads input data to the Hadoop Distributed File System (HDFS), which in turn distributes and stores it on the computing nodes.
2. The user starts the job by specifying the MapReduce program to execute along with input-output paths and other parameters.
3. The master node sends a copy of the program along with its parameters to every computing node and starts the job.
4. Computing nodes start the Map phase first by processing data on their local storage, fetching more data from other nodes if necessary and possible (this decision is up to the master node).
5. After all Map tasks are finished, their output is sorted in a way, that for every distinct Key, a Reduce task processes all the pairs with that Key.
6. Once the Reduce phase is finished and it's output has been written back to HDFS, the user then retrieves the resulting data.

Hadoop provides a fairly straightforward implementation of the MapReduce model. In order to write a complete a MapReduce job, a programmer has to specify the following things [27]:

- A *InputFormat* class, which handles reading data from disk and converting it to Key-Value pairs for the *Map* function.
- A *Mapper* class, which contains the *map* function that accepts the Key-Value pairs from *InputFormat* and outputs Key-Value pairs for the *Reduce* function.
- A *Reducer* class with a reduce function that accepts the Key-Value pairs output from the *Mapper* class and returns Key-Value pairs.
- A *OutputFormat* class, which takes Key-Value pairs from the *Reducer* and writes output to disk.

4. RESEARCH METHODOLOGY

4.1. Research Work Flow

Figure 4.1 shows the intermediate stages that have been followed for the accomplishment of this thesis work. First, the required data is collected for network intrusion detection which is then preprocessed by encoding all the string characters to numerical value and normalized it. Then it's followed by a distributed design of the DBN based on MapReduce framework for parallel operation. Finally the system is trained and tested for the network intrusion detection on and the obtained result is evaluated based on performance metrics and necessary comparison are carried out.

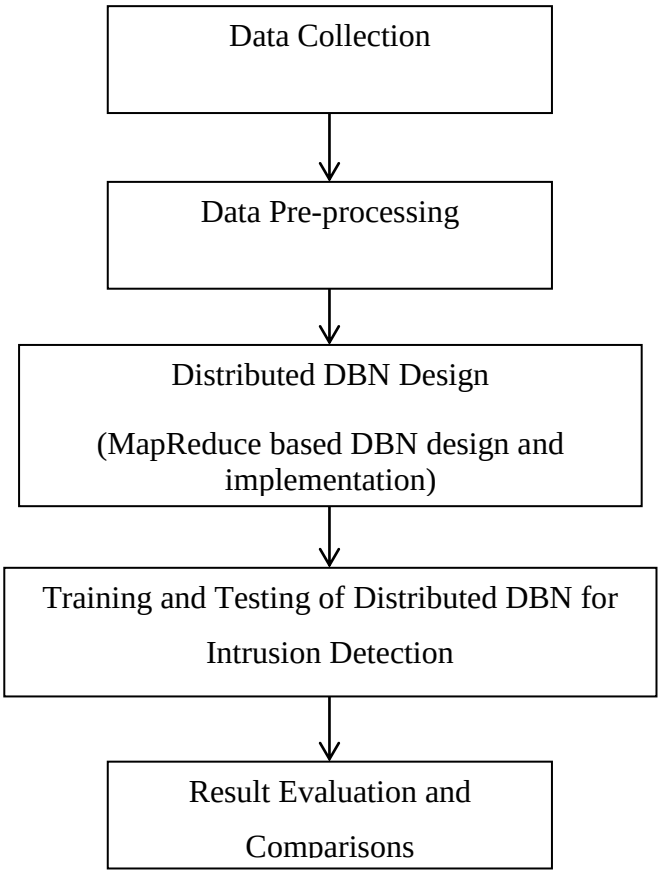

Figure 4.1: Research work flow

4.1.1. Data Collection

The two publicly available datasets UNSW-NB15 [12] and NSL-KDD[28] are used in this thesis work among many available datasets for intrusion detection. UNSW-NB15 dataset is chosen because it models the real modern normal activities and attack behavior. Whereas, the NSL-KDD is taken into consideration for benchmark comparison and it is an enhanced version of KDD'99 Cup dataset.

A) UNSW-NB15 Dataset

A UNSW-NB15 [12] is a new dataset for the evaluation of researches in network intrusion detection system created in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviors. The raw size of data is about 100 GBs and total numbers of records are 2,540,044 which are stored in the four CSV files. Each UNSW-NB15 dataset connection record contains 49 features and is labeled as either normal or an attack with one specific attack type out of 9 different attacks as shown in Table 4.1 [29].

Table 4.1: Categorization of attacks

Category	**No. of Records**	**Description**
Normal	2,218,761	Natural transaction data.
Fuzzers	24,246	Attempting to cause a program or network suspended by feeding it the randomly generated data.
Analysis	2,677	It contains different attacks of port scan, spam and html files penetrations.
Backdoors	2,329	A technique in which a system security mechanism is bypassed stealthily to access a computer or its data.
DoS	16,353	A malicious attempt to make a server or a network resource unavailable to users, usually by temporarily interrupting or suspending the services of a host connected to the Internet.

Exploits	44,525	The attacker knows of a security problem within an operating system or a piece of software and leverages that knowledge by exploiting the vulnerability.
Generic	215,481	A technique works against all block ciphers (with a given block and key size), without consideration about the structure of the block-cipher.
Reconnaissance	13,987	Contains all Strikes that can simulate attacks that gather information.
Shellcode	1,511	A small piece of code used as the payload in the exploitation of software vulnerability
Worms	174	Attacker replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it.

Table 4.2 shows the details of 49 features of UNSW-NB15 dataset and Table 4.3 reflects the part of UNSW-NB15 dataset divided into training and test set.

Table 4.2: UNSW-NB15 dataset feature details

No.	**Name**	**Type**	**Description**
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service

15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the ?ow packet size transmitted by the src
24	dmeansz	integer	Mean of the ?ow packet size transmitted by the dst
25	trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
26	res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server's http service.
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)
33	tcprtt	Float	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
34	synack	Float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
35	ackdat	Float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
36	is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0
37	ct_state_ttl	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and Post in http service.
39	is_ftp_login	Binary	If the ftp session is accessed by user and password then 1 else 0.
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.
41	ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).

42	ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
47	ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
48	attack_cat	nominal	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
49	Label	binary	0 for normal and 1 for attack records

Table 4.3: A part of the UNSW-NB15 dataset distribution

Category	**Training set**	**Testing set**
Normal	56000	37000
Analysis	2000	677
Backdoor	1746	583
DoS	12264	4089
Exploits	33393	11132
Fuzzers	18184	6062
Generic	40000	18871
Reconnaissance	10491	3496
Shellcode	1133	378
Worms	130	44
Total Records	**175341**	**82332**

B) NSL-KDD Dataset

The NSL-KDD dataset is an enhanced version of the KDD99 data set for the evaluation of researches in network intrusion detection system. Each NSL-KDD connection record contains 41 features (e.g., protocol type, service, and Flag) and is labeled as either normal or an attack with one specific attack type. These attacks fall into one of five categories listed below:

- Normal: Data with no attack.
- Denial of Service (DoS): Attacker tries to prevent legitimate users from using a service.
- Probe: Attacker tries to gain information about the target host.
- Remote to Local (R2L): Attacker does not have an account on the victim machine, hence tries to gain access.
- User to Root (U2R): Attacker has local access to the victim machine and tries to gain super user privileges.

Table 4.4 illustrates the 41 features of dataset and its type. Table 4.5 represents the distribution of dataset and only 70% of the total training set is used for the training cases in this thesis work.

Table 4.4: NSL-KDD features and its type

S.No	Feature	Feature Type	S.No	Feature	Feature Type
1	duration	continuous	22	is_guest_login	continuous
2	protocol_type	symbolic	23	count	continuous
3	service	symbolic	24	srv_count	continuous
4	flag	symbolic	25	serror_rate	continuous
5	src_bytes	continuous	26	srv_serror_rate	continuous
6	dst_bytes	continuous	27	rerror_rate	continuous
7	land	continuous	28	srv_rerror_rate	continuous
8	wrong_fragment	continuous	29	same_srv_rate	continuous
9	urgent	continuous	30	diff_srv_rate	continuous
10	hot	continuous	31	srv_diff_host_rate	continuous
11	num_failed_logins	continuous	32	dst_host_count	continuous
12	logged_in	continuous	33	dst_host_srv_count	continuous
13	num_compromised	continuous	34	dst_host_same_srv_rate	continuous
14	root_shell	continuous	35	dst_host_diff_srv_rate	continuous

15	su_attempted	continuous	36	dst_host_same_src_port_rate	continuous
16	num_root	continuous	37	dst_host_srv_diff_host_rate	continuous
17	num_file_creations	continuous	38	dst_host_serror_rate	continuous
18	num_shells	continuous	39	dst_host_srv_serror_rate	continuous
19	num_access_files	continuous	40	dst_host_rerror_rate	continuous
20	num_outbound_cmds	continuous	41	dst_host_srv_rerror_rate	continuous
21	is_host_login	continuous			

Table 4.5: NSL-KDD dataset distribution

Category	**Training set**	**Testing set**
DoS	45927	7458
U2R	52	67
R2L	995	2887
Probe	11656	2422
Normal	67343	710
Total Records	**125973**	**22544**

4.1.2. Data Preprocessing

This section describes the preprocessing process of two datasets:

A) UNSW-NB15 Dataset Preprocessing

Each record of the UNSW-NB15 training and testing dataset, which is labeled as either normal or one specific kind of attack, is described as a vector with 43 attributes with class label. Those attributes consists of continuous or discrete numerical attributes and categorical attributes. Deep belief networks require floating point numbers for the input neurons, so the numeralization of these symbolic features: proto, service and state is essential.

Using an encoding mapping method, symbolic features can be mapped to ordered numbers, as for example ‘state’ attribute has 7 different values so, it can be encoded as state type: INT=[1,0,0,0,0,0], state type: FIN =[0,1,0,0,0,0,0], state type: REQ=[0,0,1,0,0,0,0], and so on shown in Table 4.6. In the same way, symbolic features ‘proto’ with 131 distinct values, ‘service’ features with 13 distinct values and

attack_cat with 10 distinct classes can be mapped to ordered number. After successfully encoding the values from the dataset, the input numerical values have been normalized according to Equation 4.1 for keeping the values in the range of 0 and 1.

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (4.1)$$

Where, x_i = Each data point i, x_{min} = The minima among all the data points, x_{max} = The maxima among all the data points.

Finally, 43 attributes are numeralized as 190 input attributes and 10 output class attributes. Figure 4.2 represents the snapshot of input dataset after preprocessing.

Table 4.6: Numeric encoding of attribute state

State Type	**state= INT**	**state= FIN**	**state= REQ**	**state= ACC**	**state= CON**	**state= RST**	**state= CLO**
INT	1	0	0	0	0	0	0
FIN	0	1	0	0	0	0	0
REQ	0	0	1	0	0	0	0
ACC	0	0	0	1	0	0	0
CON	0	0	0	0	1	0	0
RST	0	0	0	0	0	1	0
CLO	0	0	0	0	0	0	1

	A	B	C	D	E	F	G	H	I	J	K	L
1	dur	proto=udp	proto=arp	proto=tcp	proto=igmp	proto=ospf	proto=sctp	proto=gre	proto=ggp	proto=ip	proto=ipnip	proto=st2
2	0.000011	1	0	0	0	0	0	0	0	0	0	0
3	0.000008	1	0	0	0	0	0	0	0	0	0	0
4	0.000005	1	0	0	0	0	0	0	0	0	0	0
5	0.000006	1	0	0	0	0	0	0	0	0	0	0
6	0.00001	1	0	0	0	0	0	0	0	0	0	0
7	0.000003	1	0	0	0	0	0	0	0	0	0	0
8	0.000006	1	0	0	0	0	0	0	0	0	0	0
9	0.000028	1	0	0	0	0	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0	0	0	0
11	0	0	1	0	0	0	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	0	0	0	0
13	0	0	1	0	0	0	0	0	0	0	0	0
14	0.000004	1	0	0	0	0	0	0	0	0	0	0
15	0.000007	1	0	0	0	0	0	0	0	0	0	0
16	0.000011	1	0	0	0	0	0	0	0	0	0	0
17	0.000004	1	0	0	0	0	0	0	0	0	0	0
18	0.000003	1	0	0	0	0	0	0	0	0	0	0
19	0.00001	1	0	0	0	0	0	0	0	0	0	0
20	0.000002	1	0	0	0	0	0	0	0	0	0	0
21	0.000004	1	0	0	0	0	0	0	0	0	0	0
22	0.00001	1	0	0	0	0	0	0	0	0	0	0
23	0.000009	1	0	0	0	0	0	0	0	0	0	0
24	0.00001	1	0	0	0	0	0	0	0	0	0	0
25	0.000005	1	0	0	0	0	0	0	0	0	0	0

UNSW_NB15_training-set-changed

Figure 4.2: UNSW-NB15 input dataset after preprocessing

B) NSL-KDD Dataset Preprocessing

Each record of the NSL-KDD dataset, which is labeled as either normal or one specific kind of attack, is described as a vector with 41 attributes. Those attributes consists of 38 continuous or discrete numerical attributes and 3 categorical attributes (protocol, service and flag). Same aforementioned encoding mapping and normalization method is applied to convert the categorical attributes: protocol, service and flag to numerical values. The feature 'protocol' consist 3 distinct values, feature 'service' consist 70 different values and 'flag' feature with 11 distinct values are mapped to ordered number. Thus, 41 attributes are numeralized as 122 input attributes.

4.1.3. Distributed System Design

This section describes the main design of distributed RBMs and DBNs using MapReduce. The key is to design both a Map function and a Reduce function with proper input/output key-values pairs for the MapReduce jobs.

A) Distributed RBM with MapReduce

Given an input dataset $D = \{x_i | i=1,2,...,N\}$, the goal of training an RBM is to learn the weights W, the biases b and c. In general, an iterative procedure with a number of epochs to reach convergence is necessary. In the case of distributed RBM with MapReduce, one MapReduce job is required in every epoch.

Since Gibbs sampling needs to do substantial matrix-matrix multiplications, it dominates the computation time during the training of RBM. Hence parallelizing Gibbs sampling on different data subsets in the Map phase will improve the efficiency. First, some variables are initialized such as the numbers of neurons for both visible and hidden layers, the weight W, the input layer bias b, the hidden layer bias c, the number of epochs (e.g., T) to run, and the hyper-parameters (e.g., learning rate, momentum factor). Then both the map phase and the reduce phase are repeated for T times. In each epoch, each mapper performs Gibbs sampling to compute the approximate gradients of W, b and c, and the reducer updates them with the calculated increments. (The details

for the map phase and the reduce phase are provided in the following sections.) It is noteworthy that the format of key-value pairs emitted by the reducer should be the same as that of the input for the mapper so that the output of the reducer can be as the input of the mapper in the next epoch.

Procedure 1: MapReduce job for RBM

1: Initialize the variables

2: for each epoch do

3: *Map phase*

Input: <mapID, valuelist>

Take the values and perform Gibbs sampling to compute the approximate gradients of W, b and c

Output: <key, valuelist>

4: *Reduce phase*

Input: <key, valuelist>

Sum up the approximate gradients to get the increments of W, b and c, and then update them

Output: <mapID, valuelist>

5: end for

Output: the learned W, b and c

1) Map Phase

For each mapper, the corresponding mapper ID (a number) is as the input key and the input value is a list of values. Each of the values has two elements: the first is a string (e.g., 'W') identifying the type of this value, the second is the corresponding data (e.g., it can be an M×N matrix if the first element is 'W'). In every epoch (except the first

one), the value is the output of the reducer in the previous epoch, which is the updated W, b and c and their accumulated approximate gradients. The input dataset D is divided into a number of disjoint subsets which are stored as a sequence of files (blocks) on Hadoop Distributed File System (HDFS). After reading all of the key-value pairs, each mapper loads one subset from the HDFS into memory. Given the information, each mapper can compute the approximate gradients of the weight and biases by going through all the mini-batches of the subset of the training dataset. Each mapper will emit three types of intermediate keys: *delta_W*, *delta_b* and *delta_c* which represent the increments of W, b and c, respectively, and the intermediate values have three elements: the value of *delta_W*, *delta_b* or *delta_c*, the corresponding increment and the current epoch index.

Procedure 2: The mapper of restricted Boltzmann machine (RBM) training part in deep learning MapReduce program.

Initialization:

Input of the mapper is one training case from network input. Also, there are arguments like numV is, numHid and Weight(current) past in via configurations.

Iteration:

```
1: nitialize();
2: for each data batch do
3:      getposphase();
4:      getnegphase();
5:      update();
6:      for i ← 0 to numVis−1 do
7:           for j ← 0 to numHid−1 do
8:                output <key,value> pair: <WeightID,WeightUpdate>
9:           end for
10:     end for
11: end for
10: return
```

Each mapper output its update of Weights() according to its train case.

2) Reduce Phase

For the training of RBM, there are three reducers in ideal case. Each reducer reads as input one type (i.e., *delta_W, delta_b* or *delta_c*) of the intermediate key-value pairs, and applies the reduce function to first calculate the increments and then update parameter. The reducer takes the mapper ID as the output key, and the resulting increment and the updated parameter as the output value.

Procedure 3: The reducer of restricted Boltzmann machine (RBM) training part in deep learning MapReduce program.

Initialization:

Input of the reducer is the intermediate data output by the mappers from the same weight ID.

Iteration:

1: sum ← 0;

2: for all Weightupdate ∈ same WeightID do

3: sum ← sum + Weightupdate;

4: end for

5: output <key,valuelist> pair: <WeightID,sum>

6: return

Each reducer output the overall update of Weights ().

B) Distributed DBN with MapReduce

Considering a DBN with H hidden layers, the training of this distributed DBN consists of learning H distributed RBMs for the pre-training and one distributed back-propagation algorithm for fine-tuning the global network. In addition, a main controller is required to manage the entire learning process.

1) Distributed RBMS For Pre-training

The bottom-level RBM is trained in the same way as that described in Section 4.1.3 A.The training of the rest level RBMs is also similar to the bottom-level RBM except that the input dataset is changed accordingly. The input data for the l^{th} ($H \geq l > 1$) level RBM will be the conditional probability of hidden nodes computed in the $(l-1)^{th}$ level RBM, that is

$$P(h1|x), \text{ when } l = 2;$$

$$P(h_l|h_{l-1}), \text{ when } H \geq l > 2.$$

Thus, the details of both the map function and the reduce function are omitted here.

2) Distributed Back-propagation Algorithm For Fine-tuning

In the completion of pre-training of all the hidden layers, it is time to gain discriminative power by simply putting the label layer on top of the network and iteratively tuning the weights of all the layers (i.e., $W_1 ... W_{H+1}$). Actually, in the first few epochs, the weight W_{H+1} connecting the H hidden layer and the output layer is first fine-tuned, so that it has a reasonable initialization. Note that during the fine tuning the 'weight' of each layer means the concatenation of the original weight and the bias. For the distributed back-propagation based fine-tuning, the feed-forward and back-propagation procedure, to compute the gradient of weights using gradient descent is dominated the computation time. Thus, in each epoch, this procedure is executed parallel on each subset of the data in the map phase, and then the reducers compute the weight increments and update the weights.

First load the pre-trained weights $W_1,...,W_H$ and initialize the variables such as the weight W_{H+1} and some hyper-parameters. In the map phase, each mapper takes the mapper ID as the input key, and the weight and its increment as the input value. For each data batch, the mappers calculate the gradient of weights and update the weight increments. Finally, each mapper emits the intermediate key-value pairs. In the reduce phase, each reducer takes one or more type of weights, computes the weight increments, updates the weight, and then passes back to the mappers. In the final epoch, the reducers save the fine-tuned weights, which are the final output.

Procedure 4: Fine-tuning using backpropagation

1: Load the learned weights W_1,…., W_H during the pre-training and initialize the variables

2: for each epoch do

3: *Map phase*

Input: <mapID, valuelist> pairs

Parse *valuelist* into W_1,…., W_{H+1}, *delta*_W_1,…, *delta*_W_{H+1}, and *t*

for each data batch do

Feedforward and back-propagation to get the gradients of the weights of all layers using gradient descent

Update *delta*_W_1,… *delta*_W_{H+1}

end for

Output: Emit intermediate key-value pairs <WeightID, WeightUpdate>

4: Reduce phase

Input: intermediate <key, valuelist> pairs

Compute the increment of the weight:

*delta*_W_i = sum(WeightUpdate)

Update W_i: W_i = W_i - delta_Wi

Save the learned W_i

5: end for

Output: the fine-tuned W_1,…, W_{H+1}

3) Main Controller Design

In this section, a main controller is further designed to manage the entire learning process of a DBN. The main controller schedules the running of MapReduce jobs for each level RBM and the fine-tuning.

For the first level RBM, the input data will be the training dataset D, and the pre-trained weight W1 and bias c1 are saved for loading in the fine-tuning stage. For the other RBM levels, the input data will be P ($h_{l-1}|h_{l-2}$). MapReduce jobs for the distributed back-propagation based fine-tuning. Then the pre-trained weights and biases of all levels of RBM are loaded. The resulting weights and biases of all layers are saved as the final output.

Procedure 5: The Driver for MapReduce-based Deep Learning algorithm (Main Controller)

Input: training dataset D, number of RBM levels *H*

1: for each $l \in 2$ [1;*H*] do

2: if $l == 1$ then

3: Setup for the first level RBM:

Set the training dataset D as the input data. Set the number of input neurons, the number of hidden neurons, number of epoch to train, and hyper-parameters

4: Invoke Procedure 1: RBM MapReduce

5: Save the learned weight W1, bias c1, and $P(h_1 \mid x)$

6: else

7: Setup for other level RBM:

Set $P(h_{l-1} \mid h_{l-2})$ where $h_0 = x$, as the input data. Set the number of input neurons, the number of hidden neurons, number of epoch to train, and hyper-parameters

8: Invoke Procedure 1: RBM MapReduce

9: Save the learned weight W_l, bias c_l, and $P(h_l \mid h_{l-1})$

10: end if

11: end for

12: Setup for fine-tuning:

Set the training dataset D and the corresponding labels as the input data. Load the pretrained weights and biases of all RBM levels. Set the number of epoch to train, and hyper-parameters

13: Invoke Procedure 4: fine-tunning

14: Save the final weights and bias of all layers

15: Exit

4.1.4. Performance Evaluations Metrics

In general, the performance of IDS is evaluated in term of accuracy (AC), detection rate (DR) or precision, recall, and false alarm (FA) as in the following formulas:

1. Accuracy: It is the percentage of all normal and anomaly instances that are correctly classified.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (4.2)$$

2. Detection Rate: It is rate of total anomaly detected out of total packets flow in the system. The precision is the fraction of correctly classified attacks to all attack records.

$$Detection\ Rate\ (DR) or\ Precision = \frac{TP}{TP + FP} \qquad (4.3)$$

3. Recall: The recall is the fraction of correctly classified attacks to the number of correctly classified attacks and misclassified attacks.

$$Recall\ or\ TP\ Rate\ or\ Sensitvity = \frac{TP}{TP + FN} \qquad (4.4)$$

4. False Positive Rate: It is the percentage of normal instances incorrectly classified as anomaly.

$$False\ Alarm(FA) = \frac{FP}{TN + FP} \qquad (4.5)$$

Where,

True Positive (TP) = Attacks that are correctly detected as attack

True Negative (TN) = Normal data that are correctly detected as normal

False Positive (FP) = Normal data that are incorrectly detected as attack

False Negative (FN) = Attack that are incorrectly detected as normal

The developed distributed DBN system is evaluated against the speed up and scalability with the increment of cluster size.

4.2. System Operation

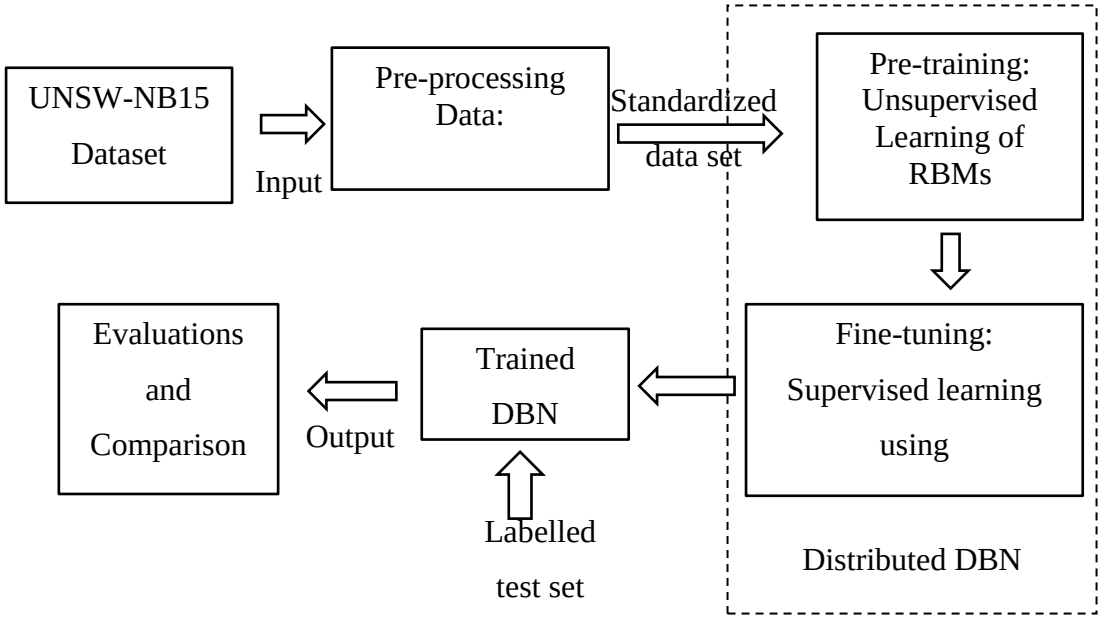

Figure 4.3: Intrusion detection system intermediate operations

As shown in Figure 4.3, the system consists of pre-training, fine tuning and evaluation and comparison sections. In pre-training section, there is a data input explained in Section 3.2.2. Each preprocessed network packet attributes are fed into the input layer of the Deep Belief Network (DBN) where first layer RBM takes it as visible units. There are 190 dimensional feature vectors in each preprocessed data. So the number of inputs of the first RBM as visible units is 190 resulting 190 input neurons. The constructed DBN is stack of 3 hidden layer of RBM and one logistic layer at output layer. First RBM samples the each input data using Equation 3.4 and sampled data is again back to visible units for re-sampling at visible units using Equation 3.5. Then, the parameters: weight between visible and hidden, visible unit bias and hidden unit bias are adjusted. The output of the first RBM is representation of feature detected from the input network packet data in probabilistic way using CD-1 algorithm as explained in Section 3.1.2. First sampled output from the first RBM is again fed into another RBM as input. Second RBM again samples it for further abstract feature dectection in heirachical fashion. In this way , pretraing is being proceed for all RBM in the system which is done in unsupervised manner. After finishing the pre-training, the system goes to fine-tuning section. In fine tuning section, the system trains the DBN in supervised way with labeled data. The labeled data which is a simple two dimension matrix where row denotes the network packet instance and column represents the 10 different attack categories. ‘1’ at the output denotes the specific attack categroy of intrusion where as ‘0’ denotes no occurrence of specific attack. The prediction is found at using feed forward calculation and then prediction output and actual output are compared for error calculation. The error obtained is back- propagated for parameter adjustment so that the accuracy is high.

For testing, the preprocessed network packets of corresponding feature vector are fed at input and prediction result is recorded. Finally, the prediction result with highest prediction value is categorized as predicted attack class.

4.3. Experimental Setup and Tools

The datasets used for this this research are UNSW-NB15 [12] and NSL-KDD [28]. All the experiments were done in Amazon EC2 cloud-computing service. The AWS cloud-computing platform is built up by multiple EC2 instances. Up to 11 EC2

instances have been used in this thesis experiments for comparing the performance of experiments in different running instances. Massive input data and intermediate results are stored in the distributed cache offered by the platform. Each EC2 instances has Intel Xeon E5-2666 v3 processors @ 2.6 Ghz, 3.75 GB memories and high network performance. Moreover each node can be boosted up to 2 virtual CPU with the performance increased. Figure 4.4 shows the 11 number of EC2 instances in Amazon cloud-computing service. Open source framework Hadoop is adopted by AWS for the distributed architecture of those nodes. The cluster is configured with Hadoop 2.7.3 with Amazon EMR release label emr-5.8.0 as shown in Figure 4.5.

The experiment other than MapReduce based DBN are conducted on a system with specification: Intel core i7 7^{th} generation @ 3.60 GHz, 16 GB RAM, NVidia GeForce 1080 GTX 8 GB DDR5 GPU. The tools and software's that are used in this thesis work are listed below:

- Hadoop 2.7.3 for MapReduce framework implementation.
- Java 1.8.121
- Eclipse Neon v2
- VMware Workstation Pro
- Python 3.5.2
- Microsoft Word, Excel
- EndNote
- PuTTy: It is a free and open-source terminal emulator; serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. In this thesis it is used for the SSH connection to AWS cluster instances.
- Scikit-learn v0.19.0: Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings [30].

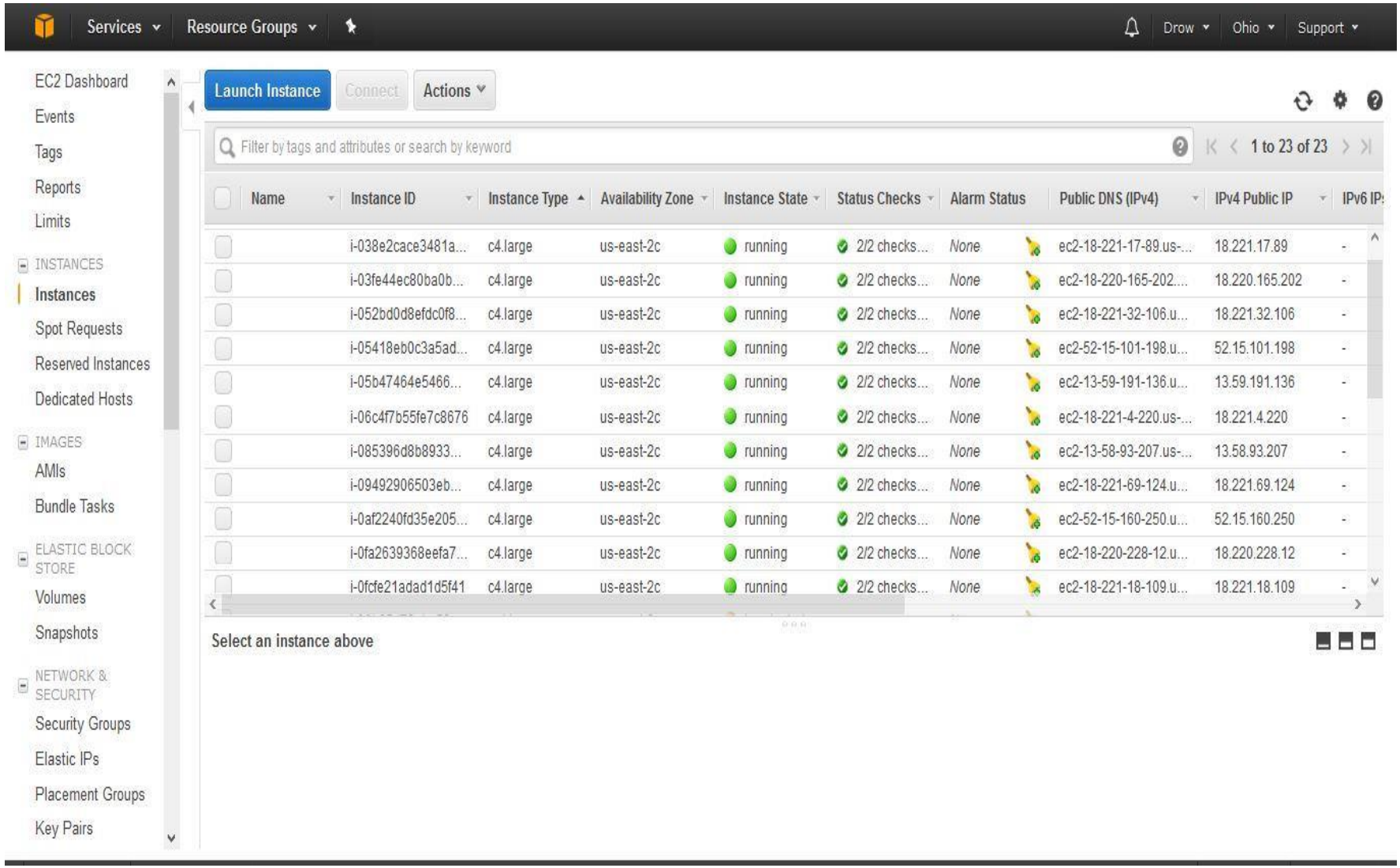

Figure 4.4: Instances (nodes) running in AWS cloud computing platform

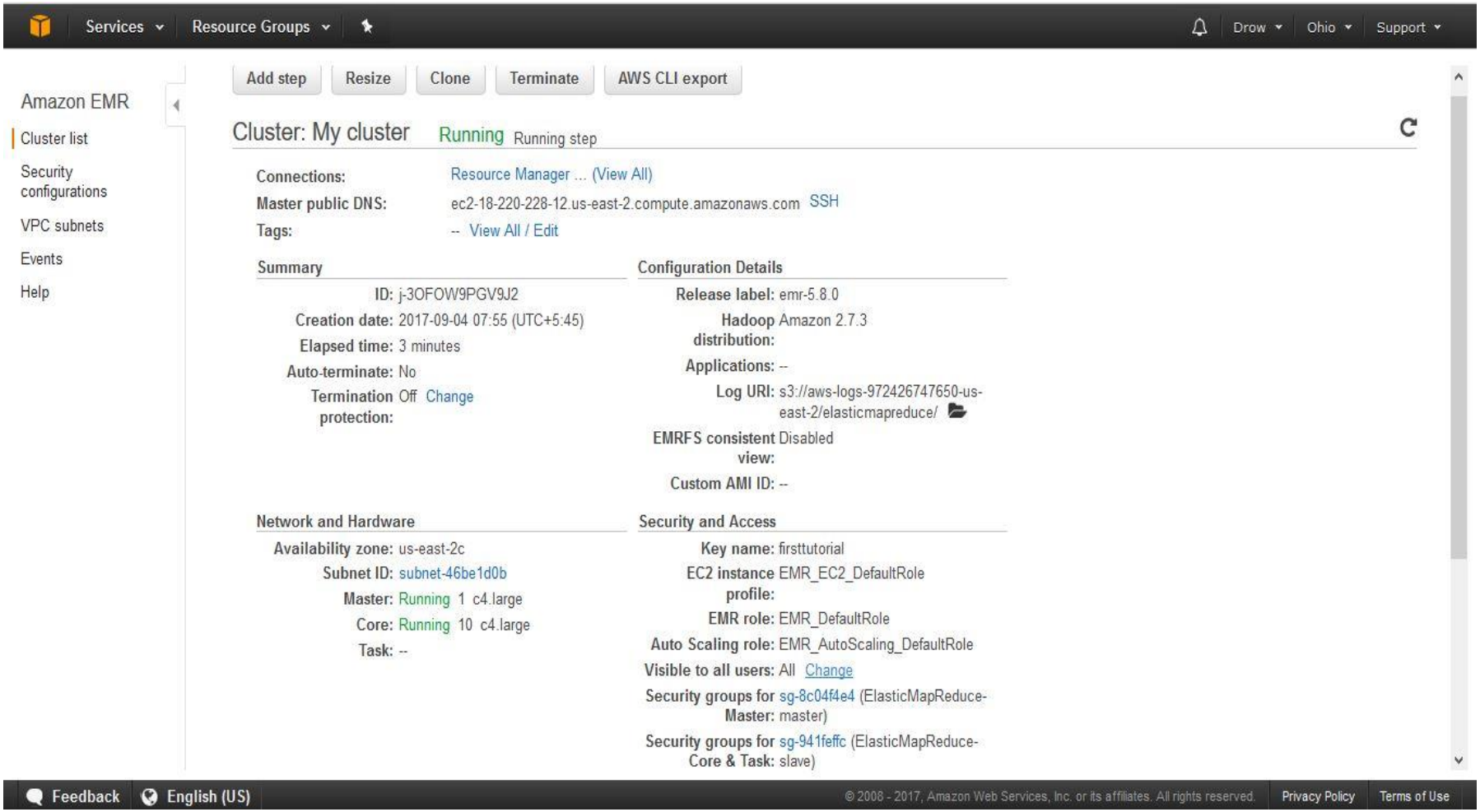

Figure 4.5: Amazon Elastic MapReduce cluster information dashboard

5. RESULTS, ANALYSIS AND COMPARISONS

This section describes the obtained result from the experiments and its analysis and comparisons. UNSW-NB15 network intrusion dataset is used as a primary dataset for system training, testing, analysis and comparison. In addition, NSL-KDD dataset is also used for the benchmark performance comparison.

5.1. Scalability and Reconstruction Error Test of System

First, the distributed DBN implemented in MapReduce framework is tested for the scalability. In this case the architecture and hyper parameter of Experiment 2 (shown in Table 5.1) is used for DBN. Three different Hadoop cluster with: 4 Datanode, 6 Datanode and 10 Datanode are used to compute the training time for different UNSW-NB15 dataset sample input. It is clear from Figure 5.1 that initially for low number of sample data, the difference in training time required is not much significant but as the number of samples increases the required training time decreases significantly for the higher cluster size. The training speed for 10 nodes cluster is 2.2 times faster than that of 4 nodes cluster and 1.5 times faster than 6 nodes cluster. Hence, MapReduce based Deep Belief Network (MRDBN) for intrusion detection is highly scalable whose speed performance can be increment with additional clusters (Datanode).

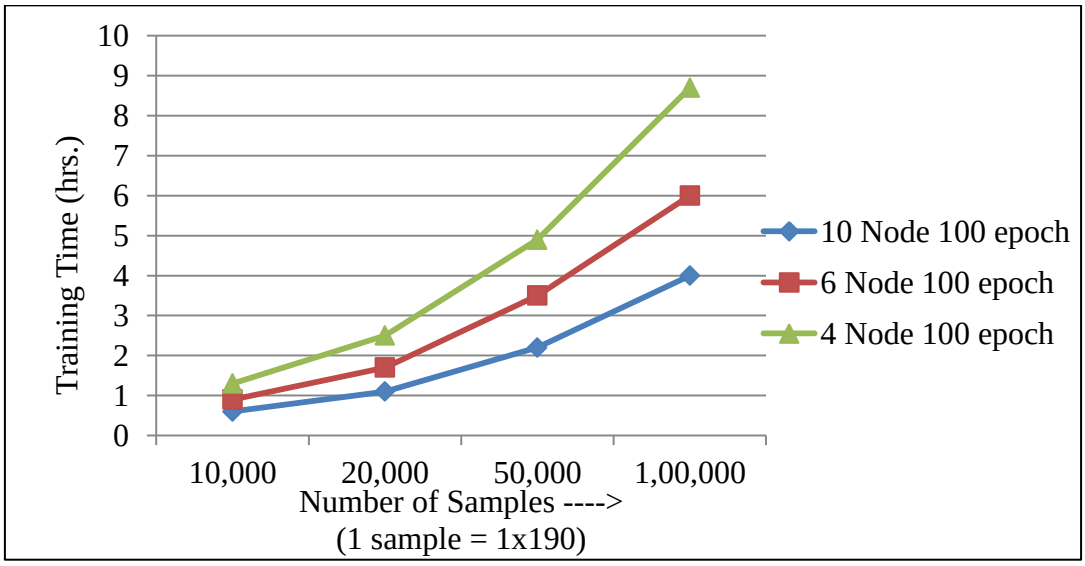

Figure 5.1: Training time versus sample for various cluster size

Second, the reconstruction error for unsupervised learning is calculated as shown in Figure 5.2. The reconstruction error for the few initial epoch decreased drastically which on later epoch it remains almost steady ending up with value around 1.5.

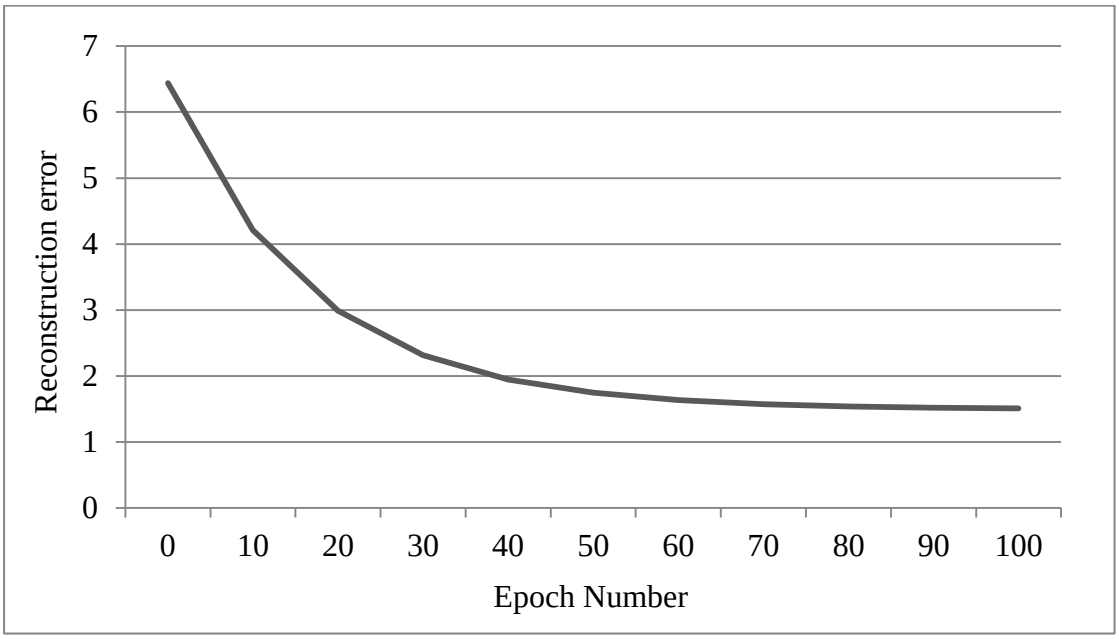

Figure 5.2: Reconstruction error vs. epoch

5.2. Performance on UNSW-NB15 Dataset

There are 3 different experiment carried out for the tuning the hyper parameter and architecture of MRDBN shown below in Table 5.1. The obtained results with corresponding parameters are listed with experiment wise below:

Table 5.1: Hyper parameters and architecture for 3 different MRDBN experiments

Parameter	**Experiment 1 Value**	**Experiment 2 Value**	**Experiment 3 Value**
No of Hidden Layers	3	3	3
Total Number of Neuron per layer	150, 90, 60	500,500,200	500, 500, 200
Input and Output Neuron	190, 10	190, 10	190, 10
Size of mini batch	50	50	50
Pre-training epoch	500	700	500
Pre-training learning rate	0.1	0.1	0.1
Fine-tuning epoch	200	200	200
Fine-tuning learning rate	0.1	0.1	0.1
Total testing sample	79879	79879	79879

The confusion matrix for the obtained for the above three experiments are shown in Table 5.2, Table 5.3 and Table 5.4

Table 5.2: Confusion matrix for experiment 1

Class\Classified as	a	b	c	d	e	f	g	h	i	j
a= Normal	26755	0	9	1689	37	13	198	19	0	2
b= Backdoor	1	66	0	30	4	1	534	75	0	0
c= Analysis	136	1	44	22	0	0	596	57	0	1
d= Fuzzers	2062	4	0	4480	37	7	898	86	0	1
e= Shellcode	30	0	0	35	297	17	70	16	0	8
f= Reconnaissance	12	2	0	7	4	3215	925	80	0	2
g= Exploits	259	10	2	166	52	225	12348	750	12	57
h= DoS	45	12	1	51	46	28	4217	675	2	22
i= Worms	0	0	0	4	0	0	20	0	28	4
j= Generic	12	5	1	21	8	3	256	49	2	17901

Table 5.3: Confusion matrix for experiment 2

Class\Classified as	a	b	c	d	e	f	g	h	i	j
a= Normal	27184	0	4	1362	15	11	135	9	0	2
b= Backdoor	2	74	3	68	4	2	372	186	0	0
c= Analysis	105	8	79	65	0	0	422	178	0	0
d= Fuzzers	2069	20	7	4563	33	14	681	183	1	4
e= Shellcode	65	0	0	48	282	6	60	10	0	2
f= Reconnaissance	16	3	0	6	4	3247	739	230	1	1
g= Exploits	241	23	8	274	46	250	11291	1712	6	30
h= DoS	45	12	5	102	35	28	3486	1369	0	17
i= Worms	1	0	0	3	0	0	42	2	6	2
j= Generic	19	5	0	19	7	1	228	86	1	17892

Table 5.4: Confusion matrix for experiment 3

Class\Classified as	a	b	c	d	e	f	g	h	i	j
a= Normal	25860	1	33	1642	35	10	170	27	1	1
b= Backdoor	2	66	0	4	3	5	595	10	0	1
c= Analysis	99	0	75	3	0	0	656	3	0	0
d= Fuzzers	2007	0	2	4299	44	8	945	21	0	2
e= Shellcode	41	1	0	38	280	10	61	24	0	1
f= Reconnaissance	16	1	0	10	11	3152	911	16	0	0
g= Exploits	265	9	11	115	63	275	12331	279	13	60
h= DoS	43	1	3	38	29	26	4311	495	1	22

i= Worms	1	2	0	1	0	0	20	3	26	2
j= Generic	14	4	0	12	3	3	227	41	0	17350

5.2.1. Evaluation and Analysis

The evaluation is performed using evaluation metrics discussed above in Section 4.1.4 for the 3 different experiments from their confusion matrix shown in Table 5.5.

Table 5.5: Prediction output summary of Experiment 1, 2 and 3

	Experiment 1	**Experiment 2**	**Experiment 3**
Recall (TP Rate)	0.824	0.826	0.827
FP Rate (FAR)	0.042	**0.039**	0.043
Precision	0.819	0.823	0.827
F-Measure	0.807	**0.818**	0.807
Accuracy	0.8239	**0.8261**	0.8271

Among three experiments, Experiment 2 outperform than remaining two. It has lowest False Alarm Rate with value 0.039 and accuracy 82.61%. From above evaluation it is concluded that increasing the number of epoch for training and number of neurons per layer increases the system performance. So architecture and hyper parameter for rest of experiment is set as according to Experiment 2. The detailed evaluation by attack class of Experiment 2 is shown in Table 5.6.

Table 5.6: Detail evaluation by attack classes of Experiment 2

Class	**Recall (TP Rate)**	**FP Rate (FAR)**	**Precision**	**Accuracy**	**F-measure**
Normal	0.946	0.05	0.914	0.946452	0.93
Backdoor	0.104	0.001	0.51	0.104079	0.173
Analysis	0.092	0	0.745	0.092182	0.164
Fuzzers	0.602	0.027	0.701	0.602376	0.648
Shellcode	0.596	0.002	0.662	0.596195	0.627
Reconnaissance	0.765	0.004	0.912	0.76454	0.832
Exploits	0.813	0.093	0.647	0.813414	0.721
DoS	0.268	0.035	0.345	0.268484	0.302
Worms	0.107	0	0.4	0.107143	0.169
Generic	0.98	0.001	0.997	0.979954	0.988

5.2.2. Comparison with ANN

The result of MRDBN for intrusion detection is compared with performance of ANN. The output metrics of ANN with 4 layers: 1 input layer, 2 hidden layers and 1 output layer is shown below in Table 5.7 and Table 5.8:

Table 5.7: Confusion matrix of ANN output prediction for UNSW-NB15

Class\Classified as	a	b	c	d	e	f	g	h	i	j
a= Normal	27592	0	0	47	0	0	1062	1	0	20
b= Backdoor	70	0	0	51	0	0	590	0	0	0
c= Analysis	17	0	0	42	0	0	798	0	0	0
d= Fuzzers	6426	0	8	193	0	0	920	8	0	20
e= Shellcode	431	0	0	42	0	0	0	0	0	0
f= Reconnaissance	2695	0	0	438	0	0	1104	1	0	9
g= Exploits	1256	0	0	630	0	0	11963	0	0	32
h= DoS	346	0	1	139	0	0	4576	0	0	37
i= Worms	14	0	0	7	0	0	35	0	0	0
j= Generic	143	0	0	27	0	0	344	0	0	17744

Table 5.8: ANN prediction performance on UNSW-NB15

	ANN Output Performance
Re-call (TP Rate)	0.72
FP Rate (FAR)	0.107
Precision	0.59
F-Measure	0.639
Accuracy	0.7197

Hence, from the performance comparison between simple ANN and MRDBN for intrusion detection using UNSW-NB15 dataset presented in Figure 5.3, the DBN based intrusion detection has outperformed ANN with overall accuracy 82.61% while ANN's accuracy is 71.97%. Moreover, the False Alarm rate for MapReduce based DBN is very less than ANN.

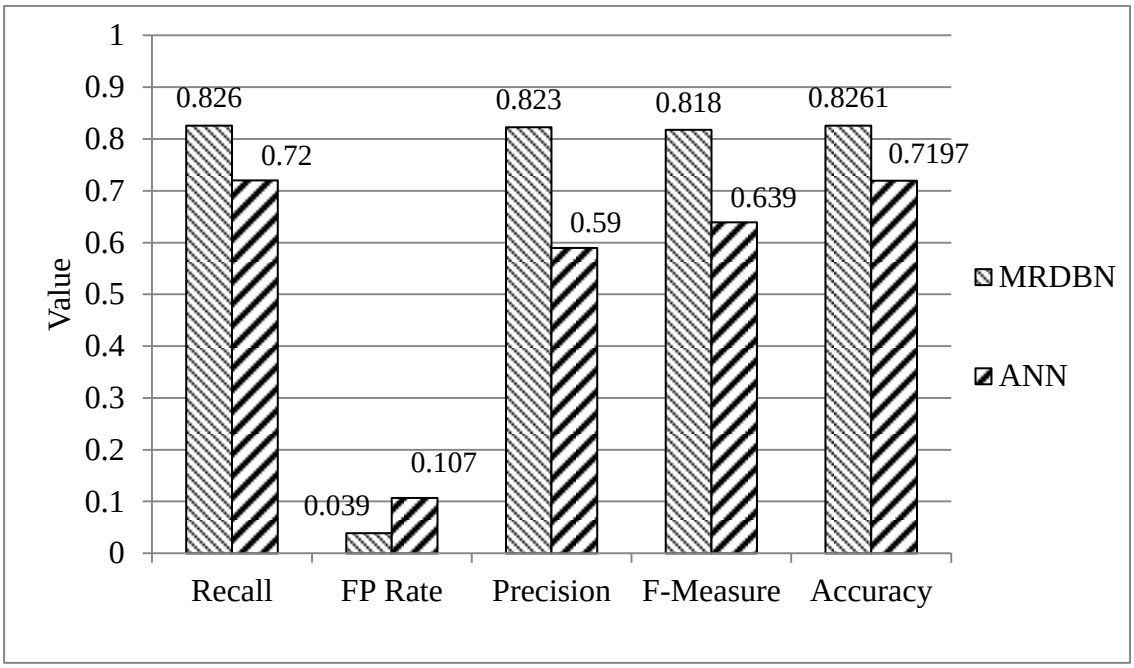

Figure 5.3: Performance comparison between MRDBN and ANN for UNSW-NB15

5.2.3. Comparison with SVM

Here the result of MRDBN for intrusion detection in UNSW-NB15 dataset is compared with SVM. The output metrics of SVM with linear kernel is given below in Table 5.9.

Table 5.9: Confusion matrix for SVM output prediction for UNSW-NB15

Class\Classified as	a	b	c	d	e	f	g	h	i	j
a= Normal	27105	0	1	638	0	473	505	0	0	0
b= Backdoor	42	0	0	8	0	29	623	9	0	0
c= Analysis	121	0	15	0	0	0	708	13	0	0
d= Fuzzers	4432	0	0	1672	0	401	1038	15	0	17
e= Shellcode	170	0	0	32	0	269	2	0	0	0
f= Reconnaissance	478	0	0	77	0	2433	1238	11	0	10
g= Exploits	1150	0	0	308	0	268	12002	117	0	36
h= DoS	214	0	0	66	0	98	4633	51	0	37
i= Worms	5	0	0	4	0	4	43	0	0	0
j= Generic	60	0	0	64	0	37	349	0	0	17748

Table 5.10: SVM prediction performance on UNSW-NB15

	SVM Output Performance
Re-call (TP Rate)	0.764
FP Rate (FAR)	0.074
Precision	0.727
F-Measure	0.719
Accuracy	0.7639

Hence, the comparative analysis between simple SVM and MRDBN for intrusion detection using UNSW-NB15 dataset is presented in Figure 5.4. The MRDBN based intrusion detection has overcame SVM with overall accuracy 82.61% while SVM's accuracy is 76.39%. Moreover, the False Alarm rate for MapReduce based DBN is quite less than SVM.

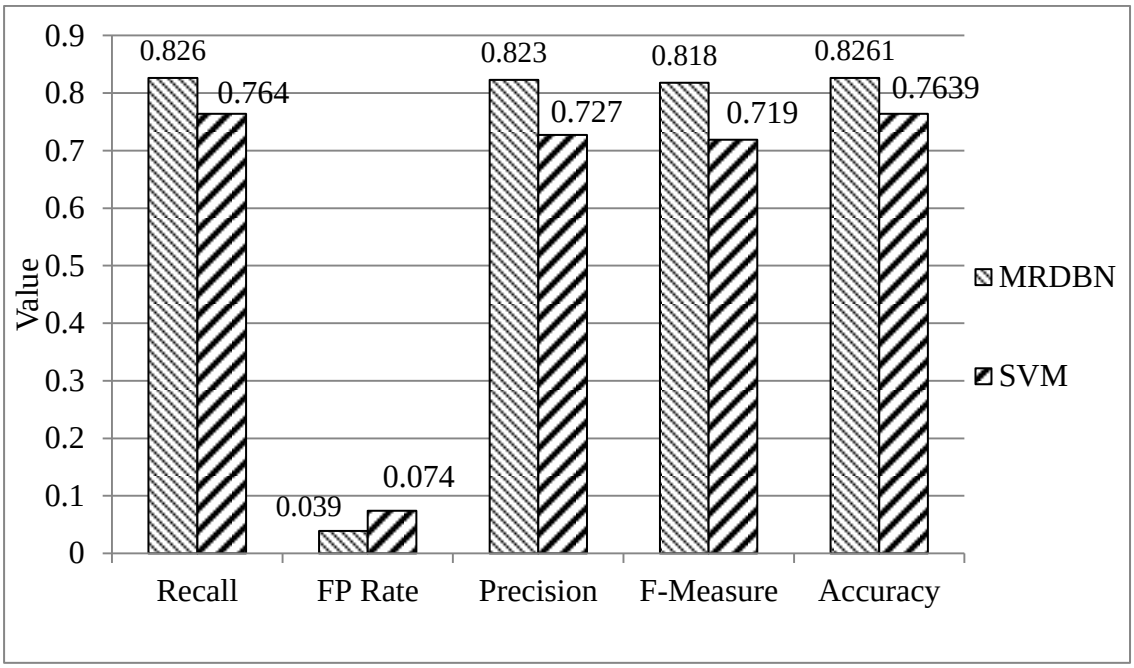

Figure 5.4: Performance comparison between MRDBN and SVM for UNSW-NB15

5.2.4. Comparison between MRDBN, SVM and ANN

Figure 5.5 exemplifies the detailed comparative analysis for intrusion detection using MRDBN, SVM and ANN for UNSW-NB15 dataset. It is vivid that accuracy of MRDBN is more than SVM and SVM is more than ANN. On the contrary, False Alarm rate of MRDBN is very less than SVM and ANN.

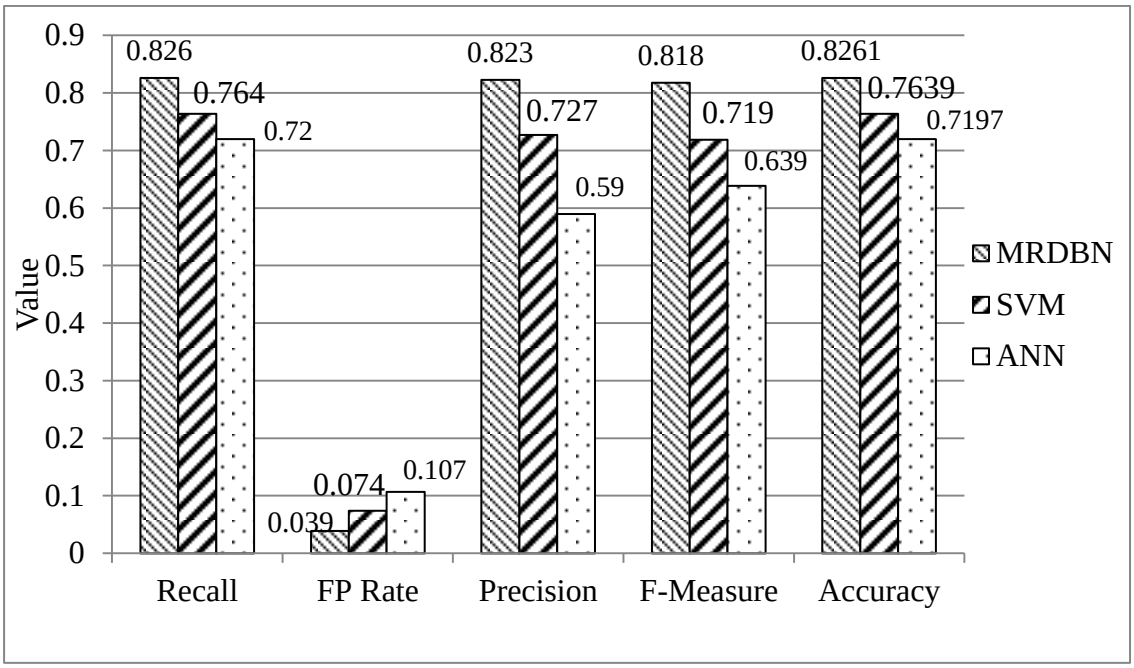

Figure 5.5: Comparative Analysis of MRDBN, SVM and ANN for UNSW-NB15

Figure 5.6 shows the comparison among MRDBN, SVM and ANN by 10 attack class. The accuracy for attack class: normal, exploits and generic is almost same for above mentioned 3 algorithms. On the other hand, for remaining 7 attacks MRDBN has outperformed the prediction performance than SVM and ANN.

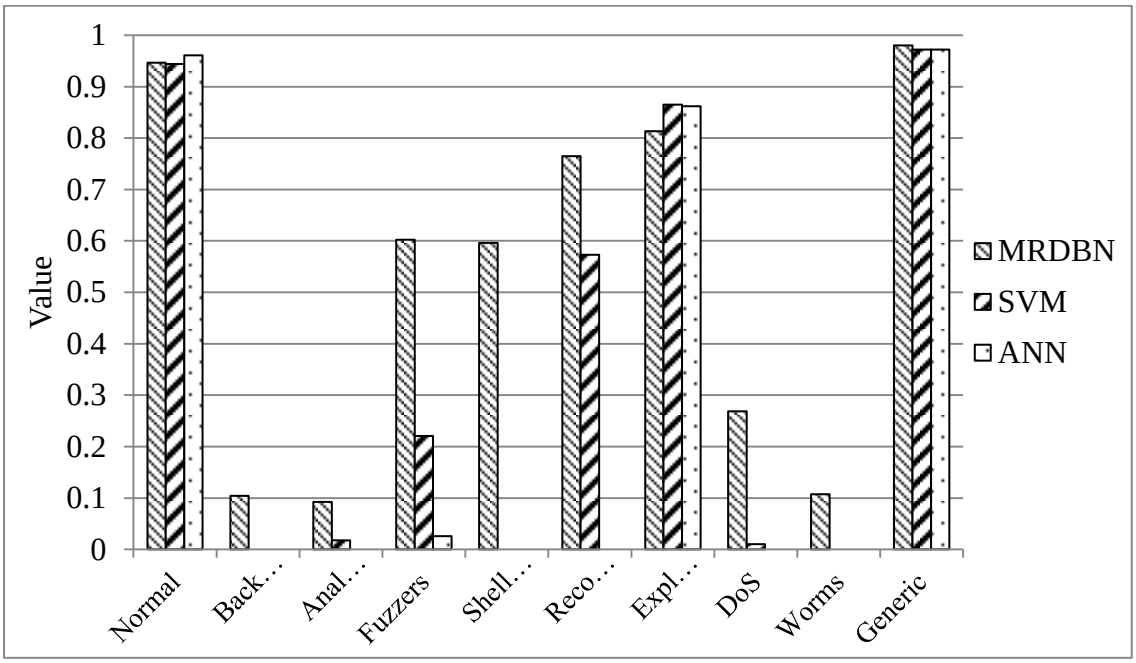

Figure 5.6: Accuracy comparison by attack class of MRDBN, SVM and ANN for UNSW-NB15

5.2.5. MRDBN Binary Classification on UNSW-NB15

Furthermore, the MRDBN for Experiment 2 setup is conducted for the binary classification of UNSW-NB15 (i.e. either the connection is “Normal” or “Attack”). The confusion matrix and output performance is shown below in Table 5.11 and Figure 5.7. The overall accuracy achieved is 94.94% and false alarm rate is 5.6%.

Table 5.11: Confusion matrix of MRDBN binary classification for UNSW-NB15

Class\ Classified as	**a**	**b**
a= Normal	26885	1837
b= Attack	2199	48958

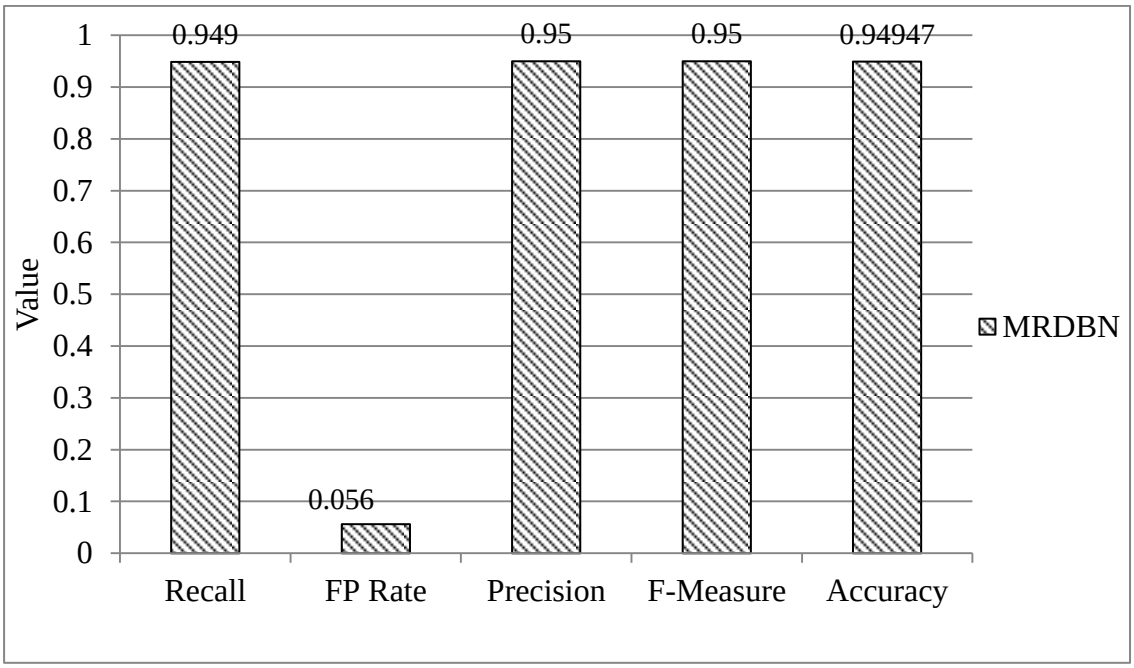

Figure 5.7: MRDBN prediction output performance for binary classification of UNSW-NB15

5.3. Performance on NSL-KDD Dataset

The performance of the developed MRDBN is also evaluated in NSL-KDD with total 22277 number of testing instances for benchmark comparison. The confusion matrix of MRDBN for NSL-KDD using DBN architecture as described above in Experiment 2 is shown below in Table.

Table 5.12: Confusion matrix of MRDBN output prediction for NSL-KDD

Class\Classified as	a	b	c	d	e
a= Normal	12276	306	2	711	6
b= Dos	34	5550	2	79	0
c= R2L	1026	0	1144	7	2
d= Probe	2	22	0	1072	0
e= U2R	27	0	1	0	8

5.3.1. Evaluation and Analysis

The detailed evaluation of system by attack class type is shown in Table 5.13 and Table 5.14 shows the overall performance metrics for NSL-KDD using MRDBN.

Table 5.13: Detail evaluation by attack classes of MRDBN for NSL-KDD

Class	TP Rate (Recall)	FP Rate	Precision	F-measure
Normal	0.923	0.121	0.919	0.921
Dos	0.98	0.02	0.944	0.962
R2L	0.525	0	0.996	0.688
Probe	0.978	0.038	0.574	0.723
U2R	0.222	0	0.5	0.308

Table 5.14: MRDBN prediction performance on NSL-KDD

	MRDBN Output Performance
Re-call (TP Rate)	0.9
FP Rate (FAR)	0.079
Precision	0.915
F-Measure	0.898
Accuracy	0.9

Above result shows that TP rate for U2R is very less in comparison to remaining attacks. The reason behind this is little number of training cases for U2R present in the dataset. The overall accuracy thus obtained is 90.03% with 7.9% of false alarm rate.

5.3.2. Comparison between MRDBN, SVM and ANN

The result of MRDBN for intrusion detection using NSL-KDD dataset is compared with performance of ANN and SVM. The confusion matrix for ANN and SVM are shown below in Table 5.15 and Table 5.16 respectively. The Table 5.17 represents the output metrics for both ANN and SVM.

Table 5.15: Confusion matrix of ANN output prediction for NSL-KDD

Class\Classified as	**a**	**b**	**c**	**d**	**e**
a= Normal	11766	328	68	1036	103
b= Dos	824	4552	1	240	48
c= R2L	1093	0	1021	1	64
d= Probe	1	0	0	1095	0
e= U2R	10	0	3	0	23

Table 5.16: Confusion matrix of SVM output prediction for NSL-KDD

Class\Classified as	**a**	**b**	**c**	**d**	**e**
a= Normal	11497	1002	37	756	9
b= Dos	23	5592	0	50	0
c= R2L	928	0	1246	4	1
d= Probe	8	1	0	1087	0
e= U2R	31	0	3	0	2

Table 5.17: ANN and SVM prediction performance on NSL-KDD

	ANN	**SVM**
Re-call (TP Rate)	0.829	0.872
FP Rate (FAR)	0.137	0.083
Precision	0.864	0.889
F-Measure	0.832	0.871
Accuracy	0.8285	0.8719

Figure 5.8 depicts the detailed comparative analysis for intrusion detection using MRDBN, SVM and ANN for NSL-KDD dataset. In this case also it is vivid that an accuracy of MRDBN is more than SVM and SVM is more than ANN. On the contrary, False Alarm rate of MRDBN is very less than SVM and ANN.

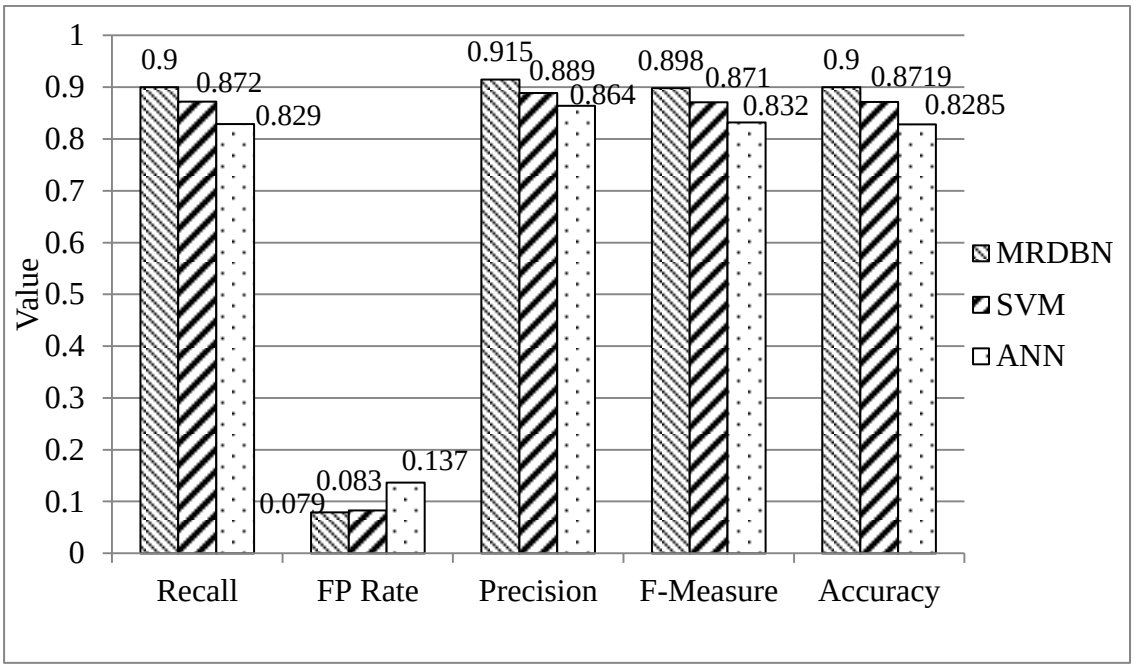

Figure 5.8: Comparative Analysis of MRDBN, SVM and ANN for NSL-KDD

Figure 5.9 shows the comparison among MRDBN, SVM and ANN by 5 attack class. The accuracy for attack class: normal, R2L and probe is almost same for above mentioned 3 algorithms. However the accuracy of ANN for U2R is significantly higher than remaining two algorithms.

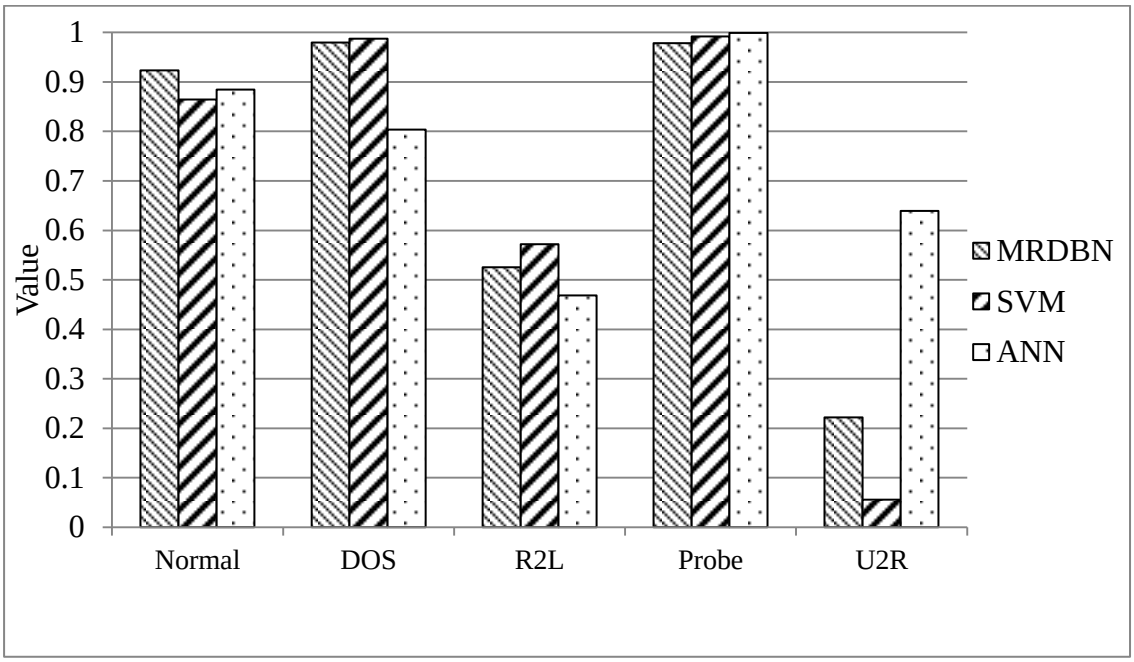

Figure 5.9: Accuracy comparison by attack class of MRDBN, SVM and ANN for NSL-KDD

5.3.3. MRDBN Binary Classification on NSL-KDD

Moreover, the MRDBN for Experiment 2 setup is conducted for the binary classification of NSL-KDD (i.e. either the connection is “Normal” or “Attack”). The confusion matrix and output performance is shown below in Table 5.11 and Figure 5.7. The overall accuracy achieved is 91.05% and false alarm rate is 9.5%.

Table 5.18: Confusion matrix of MRDBN binary classification for NSL-KDD

Class\ Classified as	**a**	**b**
a= Normal	12268	1033
b= Attack	959	8017

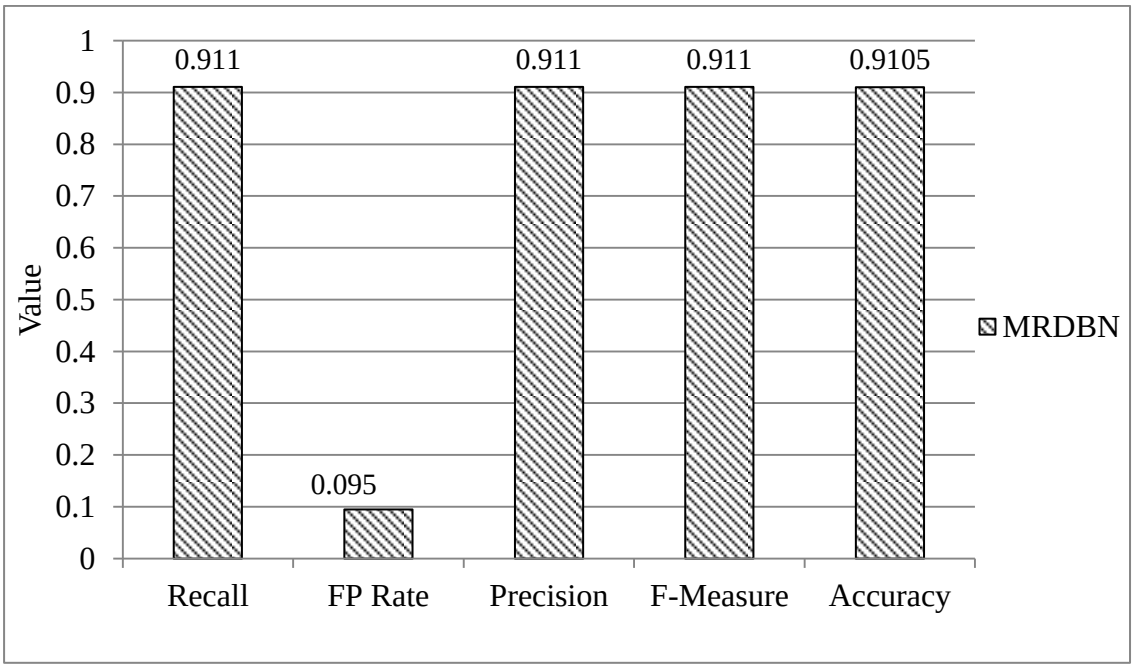

Figure 5.10: MRDBN prediction output performance for binary classification of NSL-KDD

5.4. Cloud-Computing Service Execution Results

The figures shown below are some screen shot taken during the experiments carried out in Amazon Cloud-computing service with Hadoop ecosystem. Figure 5.11 shows the method of assigning a job for Amazon EMR for executing the MapReduce job. Here DeepLearningDriver is main class for executing jar file. Then the input and output path

for data input and output is given. The number layers assigned is 5 and iteration number is 500 with 50000 sample data input.

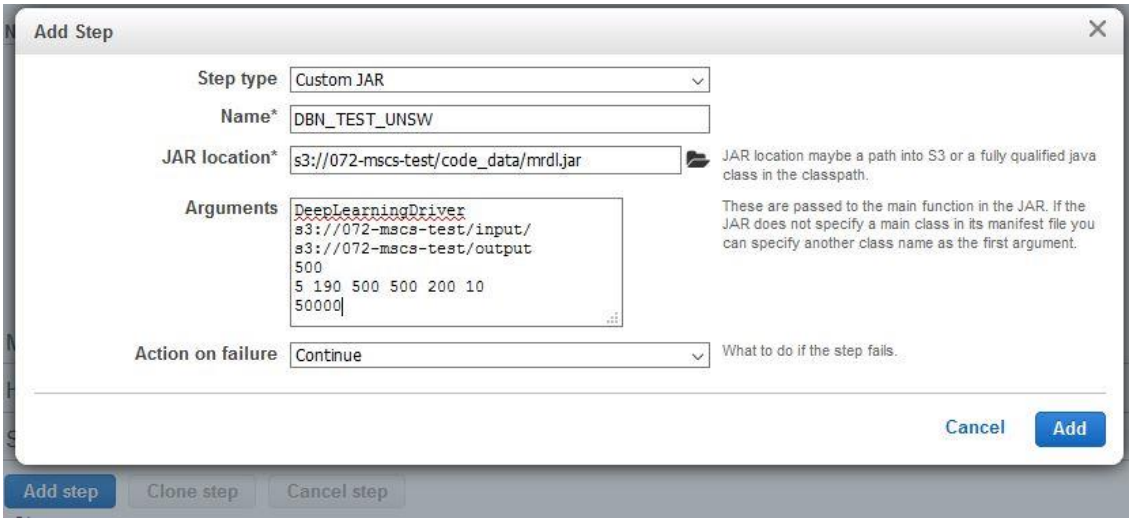

Figure 5.11: Adding step for running MapReduce job

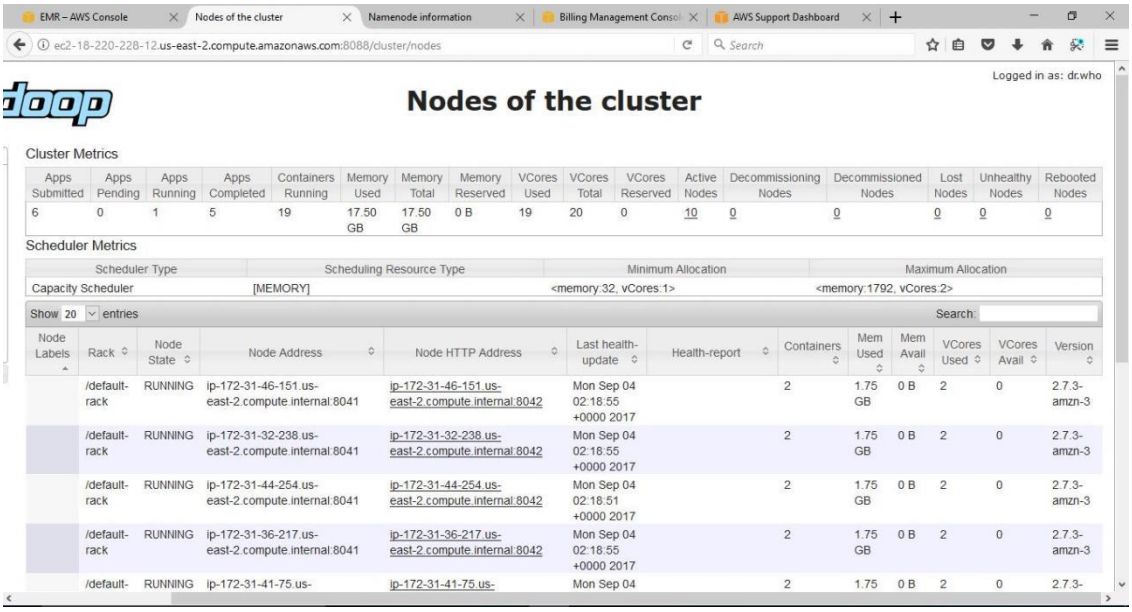

Figure 5.12: Details of Nodes running in cluster

Figure 5.12 illustrates total number of datanodes running in the cluster and details of each node. Figure 5.13 demonstrates the list of MapReduce job completed and currently in execution with start and end time.

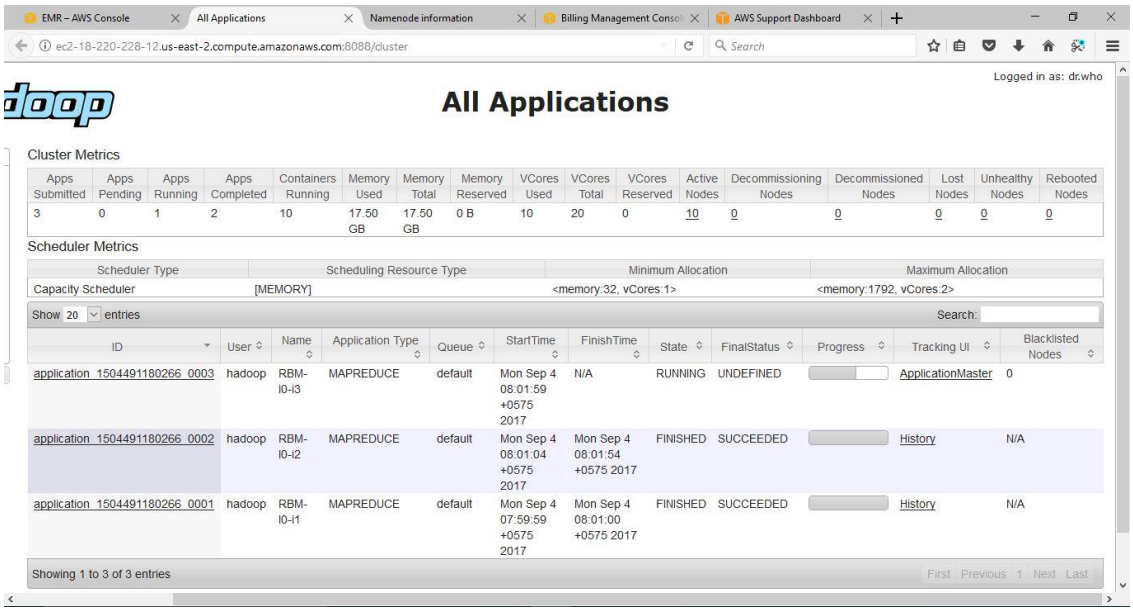

Figure 5.13: Summary of applications running in Hadoop ecosystem

Figure 5.14 depicts the Syslog information of executing the given job. It shows the details about the input path, output path, number of job split and percentage of map and reduce job completion. Also it provides the information relating number of mappers and reducers used.

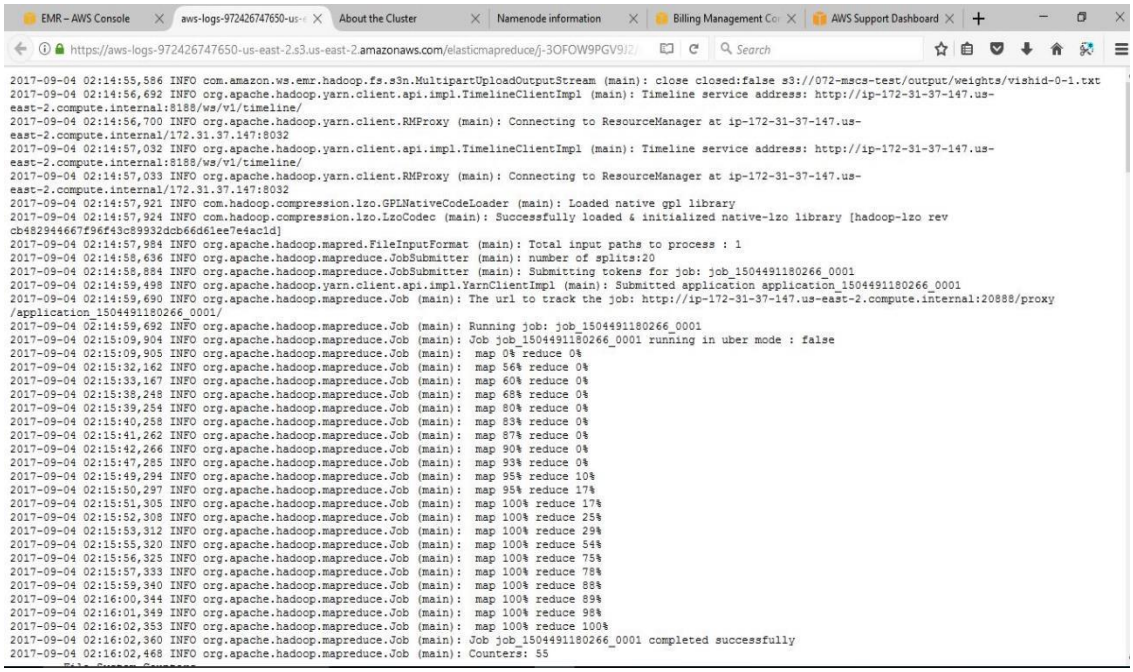

Figure 5.14: Syslog information of MapReduce job running

6. CONCLUSIONS

First, the dataset was preprocessed to convert all attributes into numerical values and normalized it. Then, the distributed DBN based on MapReduce framework was implemented. The MapReduce framework was implemented with Hadoop ecosystem in Amazon Cloud-computing service with up to 11 EC2 instances. The system consists of two phase training: pre-training with stack of RBM and fine-tuning with backpropagation. Three different model of MRDBN was tested using UNSW-NB15 dataset, among which the model of Experiment 2 performed better than remaining two. So, the model architecture of MRDBN of Experiment 2 was used for rest of remaining analysis. The training time versus cluster size shows, the 10 nodes cluster is 2.2 time faster than 4 nodes cluster. Moreover, the overall accuracy of the developed MRDBN system for multiclass classification is 82.61% with 3.9% false alarm rate for UNSW-NB15 dataset. The developed system is more precise than existing ANN and SVM. The achieved accuracy and false alarm rate for ANN is 71.97% and 10.7% false alarm rate respectively. Whereas, the SVM achieved 76.69% of accuracy with 7.4% false alarm rate. The accuracy of MRDBN for binary classification of attack is 94.94%. Moreover, the MRDBN is tested on NSL-KDD dataset which achieved an accuracy of 90.03% with 7.9% of false alarm rate. Hence, MRDBN for Intrusion Detection is highly scalable for large amount of network traffic data and more accurate than existing ANN and SVM.

Future extension of this work can be done by implementing MRDBN on Apache Spark platform for in-memory calculation thus, making it applicable for real time intrusion detection. An adaptive learning method like Neuron Generation and Annihilation Algorithm [31] can be used to discover the optimal number of hidden neurons and layers according to the input space, making the system more reliable to take an account of the computational cost and the model stability.

7. LIMITATIONS

In this thesis work, the architecture of DBN cannot be changed during the learning phase i.e. it is not adaptive. Moreover, the learning rate and the momentum take are fixed value. Training and prediction is done using batch processing method for the data stored in HDFS, so, not much reliable for real time implementation.

REFERENCES

[1] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania1980.

[2] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications,* vol. 36, pp. 11994-12000, 2009.

[3] V. Kumar, J. Srivastava, and A. Lazarevic, *Managing cyber threats: issues, approaches, and challenges* vol. 5: Springer Science & Business Media, 2006.

[4] C.-H. Tsang and S. Kwong, "Ant colony clustering and feature extraction for anomaly intrusion detection," *Swarm Intelligence in Data Mining,* pp. 101-123, 2006.

[5] E. Aminanto and K. Kim, "Deep Learning in Intrusion Detection System: An Overview," in *2016 International Research Conference on Engineering and Technology (2016 IRCET)*, 2016.

[6] A. Fischer and C. Igel, "Training restricted Boltzmann machines: An introduction," *Pattern Recognition,* vol. 47, pp. 25-39, 2014.

[7] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation,* vol. 18, pp. 1527-1554, 2006.

[8] G. E. Hinton and R. R. Salakhutdinov, "A better way to pretrain deep boltzmann machines," in *Advances in Neural Information Processing Systems*, 2012, pp. 2447-2455.

[9] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science,* vol. 313, pp. 504-507, 2006.

[10] G. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum,* vol. 9, p. 926, 2010.

[11] H. Mohamed and S. Marchand-Maillet, "MRO-MPI: MapReduce overlapping using MPI and an optimized data exchange policy," *Parallel Computing,* vol. 39, pp. 851-866, 2013.

[12] *UNSW-NB15 Dataset*. Available: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

[13] R. Heady, G. F. Luger, A. Maccabe, and M. Servilla, *The architecture of a network level intrusion detection system*: University of New Mexico. Department of Computer Science. College of Engineering, 1990.

[14] M. Moradi and M. Zulkernine, "A neural network based system for intrusion detection and classification of attacks," in *Proceedings of the 2004 IEEE international conference on advances in intelligent systems-theory and applications*, 2004.

[15] K. Faraoun and A. Boukelif, "Neural networks learning improvement using the K-means clustering algorithm to detect network intrusions," *INFOCOMP Journal of Computer Science*, vol. 5, pp. 28-36, 2006.

[16] I. Ahmad, A. Abdullah, A. Alghamdi, and M. Hussain, "Optimized intrusion detection mechanism using soft computing techniques," *Telecommunication Systems*, pp. 1-9, 2013.

[17] D. Gaikwad, S. Jagtap, K. Thakare, and V. Budhawant, "Anomaly based intrusion detection system using artificial neural network and fuzzy clustering," in *International journal of engineering research and technology*, 2012.

[18] V. Jaiganesh, P. Sumathi, and S. Mangayarkarasi, "An analysis of intrusion detection system using back propagation neural network," in *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*, 2013, pp. 232-236.

[19] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, 2014, pp. 247-252.

[20] S. Jo, H. Sung, and B.-H. Ahn, "A Comparative Study on the Performance of SVM and an Artificial Neural Network in Intrusion Detection," *Journal of the Korea Academia-Industrial cooperation Society*, vol. 17, pp. 703-711, 2016.

[21] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish, and A. E. Hassanien, "Hybrid intelligent intrusion detection scheme," in *Soft computing in industrial applications*, ed: Springer, 2011, pp. 293-303.

[22] M. Z. Alom, V. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief networks," in *Aerospace and Electronics Conference (NAECON), 2015 National*, 2015, pp. 339-344.

[23] S. Son, M.-S. Gil, and Y.-S. Moon, "Anomaly detection for big log data using a Hadoop ecosystem," in *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, 2017, pp. 377-380.

[24] X. S. Lin, B. W. Li, and X. Y. Yang, "Engine components fault diagnosis using an improved method of deep belief networks," in *Mechanical and Aerospace Engineering (ICMAE), 2016 7th International Conference on*, 2016, pp. 454-459.

[25] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1798-1828, 2013.

[26] K. Zhang and X.-W. Chen, "Large-scale deep belief nets with mapreduce," *IEEE Access*, vol. 2, pp. 395-403, 2014.

[27] T. White, *Hadoop: The definitive guide*: " O'Reilly Media, Inc.", 2012.

[28] *NSL-KDD dataset (July 2010 ed.)*. Available: http://kdd.ics.uci.edu/databases

[29] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Military Communications and Information Systems Conference (MilCIS), 2015*, 2015, pp. 1-6.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[31] S. Kamada and T. Ichimura, "An adaptive learning method of restricted Boltzmann machine by neuron generation and annihilation algorithm," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*, 2016, pp. 001273-001278.