School of Computing, Engineering and Mathematics

A thesis submitted in the partial fulfilment of the requirements for the degree of

MASTER OF RESEARCH IN COMPUTER SCIENCE

A Deep Learning Approach for Intrusion Detection in Internet of Things using Bi-Directional Long Short-Term Memory Recurrent Neural Network

By

Bipraneel Roy

December, 2018

THIS PAGE LEFT INTENTIONALLY BLANK

DEDICATION

I owe the most to my mother Bakul Rani Roy, for her genuine optimism and continuous inspiration. Without her encouragement and support this piece of work would have been nowhere. I sincerely put my gratitude towards her for supporting me in every way one can be supported and dedicate my MRes Thesis to my Mother.

THIS PAGE LEFT INTENTIONALLY BLANK

ACKNOWLEDGEMENT

It is truly an incredible delight to thank everybody who helped me in this undertaking. Firstly, I would like to convey my sincere gratitude to my supervisor Dr. Hon Cheung, for his invaluable comments, suggestions and continuous support and optimism throughout the project.

I would like to thank University of Western Sydney for giving me the fortuity for conducting Masters Course and School of Computing Engineering and Mathematics (SCEM) department for providing me with all the resources needed to accomplish it. I would conjointly wish to convey the ITNAC conference committee for accepting my paper.

Also, my sincere gratitude goes to my friend Taniya Gomes for providing moral, economic and intellectual upkeep all through the year. I likewise want to thank Niranjan Parajuli for being such a supportive friend.

Finally, I also convey my thankfulness towards those that don't seem to be listed here. So many people have helped me during my Masters study. At last, obviously, the obligation regarding any error that may remain is solely mine.

THIS PAGE LEFT INTENTIONALLY BLANK

STATEMENT OF AUTHENTICATION

The work presented in this thesis is to the best of my knowledge and believe, original except as acknowledged in the text. I hereby declare that I have not submitted this material either in full or in part, for a degree at this or any other institution.

Signature

THIS PAGE LEFT INTENTIONALLY BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREBRIATIONS

ACCS	Australian Centre for Cyber Security
AE	Auto Encoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programmer Interface
BAC	Breast Arterial Calcification
BLSTM	Bi-directional Long Short-Term Memory
BP	Back Propagation
BPTT	Back Propagation Through Time
BRNN	Bidirectional Recurrent Neural Network
CAE	Contractive Auto Encoder
CART	Classification And Regression Tree
CEL	Cross Entropy Loss
CPS	Cyber-Physical System
CPU	Central Processing Unit
DAE	Denoising Auto Encoder
DL	Deep Learning
DNN	Deep Neural Network
DOS	Denial of Service
EM	Expectation Maximization
FAR	False Alarm Rate
FDC	Fault Detection and Classification
FFNN	Feed Forward Neural Network

FN	False Negative
FP	False Positive
FPR	False Positive Rate
GAN	Global Area Network
GPRS	General Packet Radio Services
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System For Mobile
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IERC	European Research Cluster On Internet-of-Things
IOT	Internet Of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ITS	Intelligent Transportation System
ITU	International Telecommunications Union
LSTM	Long Short Term Memory
LSTM S2S	Long Short-Term Memory Sequence-to-Sequence
ML	Machine Learning
MSE	Mean Squared Error
OOP	Object oriented programming
QoS	Quality Of Service

ReLU	Rectified Linear Unit
RFID	Radio Frequency Identification
RNN	Recurrent Neural Network
SDA	Stacked Denoising Auto Encoder
SDN	Software Defined Network
SGD	Stochastic gradient descent
SVM	Support Vector Machine
TCP	Transfer Control Protocol
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
WAN	Wide Area Network
WSN	Wireless Sensor Network

ABSTRACT

Internet-of-Things connects every ‘thing’ with the Internet and allows these ‘things’ to communicate with each other. IoT comprises of innumerous interconnected devices of diverse complexities and trends. This fundamental nature of IoT structure intensifies the amount of attack targets which might affect the sustainable growth of IoT. Thus, security issues become a crucial factor to be addressed. A novel deep learning approach have been proposed in this thesis, for performing real-time detections of security threats in IoT systems using the Bi-directional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN). The proposed approach have been implemented through Google TensorFlow implementation framework and Python programming language. To train and test the proposed approach, UNSW-NB15 dataset has been employed, which is the most up-to-date benchmark dataset with sequential samples and contemporary attack patterns. This thesis work employs binary classification of attack and normal patterns. The experimental result demonstrates the proficiency of the introduced model with respect to recall, precision, FAR and f-1 score. The model attains over 97% detection accuracy. The test result demonstrates that BLSTM RNN is profoundly effective for building highly efficient model for intrusion detection and offers a novel research methodology.

Keywords—Bi-directional Recurrent Neural Network, Deep Learning, Intrusion Detection, IoT.

Chapter 1

INTRODUCTION

1.1. The Overview

The Internet, since 1960s, has been playing an important role in connecting individuals and putting organization and businesses together. It has collapsed the geographical barriers that previously used to exist between peoples and has provided an efficient and financially worthwhile way of communications.

These days, things are changing and opening a completely new dimension of communication due to the emergence of smart objects which possess the competency of creating and collaborating data through the Internet in a much smarter course. Internet of Things (IoT) is the cutting-edge innovation and frameworks which can possibly change the way in which we live. IoT can be viewed as an innovation which is built upon two fundamental components: “Internet” and “Things”. The “Things” simply refers to any kind of device or object that has the capability of perceiving or collecting information about itself or the surrounding environment. These smart devices or things has the capability of analyzing and acting accordingly with other devices by using “Internet” as the backbone network for communication.

IoT communication systems can reach way beyond the traditional Internet and has the potential to improve human life condition. For instance, through IoT, human health can be remotely monitored, thus, rejecting the necessity of visiting the hospital physically. For example, University of Edinburgh, Scotland, have created minute computing gadgets that can be attached to human chest, and can screen and gather respiratory information and after that transmit it remotely to the respective specialists who can pursue their cases remotely [114]. IoT is being utilized by government organizations round the globe for gathering information from various regions and to make accessible enhanced facilities in security, health, development and transportation. IoT is employed by enterprises for accommodating enhanced customer services and to augment security and safety to employees. IoT can also enhance the way of managing day to day life. For instance, Amazon Echo are a bunch of smart IoT devices having a linguistic capability. People can interact with the devices and can ask for advice regarding weather, schedule alarm or obtain new feed from the Internet.

Internet of Things (IoT), originally termed by Kevin Ashton in the year 1999 [2], stands for a system of globally recognizable physical devices (or things) which can sense the environment around them and behave intelligently. To construct the IoT network, a varied assortment of technologies is required. These techniques support to shape a virtual world of objects or things over the physically connected networks where things can communicate to each other in an intelligent way, providing information to people or taking actions based on individual inputs. IoT is rising at an accelerating stride, interconnecting billions of device or 'Things'. As per Gartner [1], about 25 billion distinctively recognizable objects or things are predicted to be a part of the worldwide computing system by 2020 [1]. These interconnected devices augment regular activities and shape smart solutions. However, the immense prospects and conveniences brought by IoT leads to security concerns.

1.2. Motivation

The technique of Deep Learning (DL) essentially imitates the functionality of a human brain. For acquiring those functionalities, the DL technique uses powerful NN algorithms such as Clustering algorithm, Bayesian Algorithm, Artificial Neural Network algorithm. DL algorithms possess high computation ability which makes it more appropriate for intricate and composite IoT datasets likened to legacy ML techniques. DL application in the IoT sphere, chiefly on IoT network security is still in its early research phase and holds an enormous prospective for discovering incursions from the IoT system. Recurrent neural networks possess the ability of learning from preceding time-steps from input dataset. The information of every time-step goes under processing, and then reserved for providing input for the succeeding time-stamp. The subsequent time-step uses the prior data stored for processing the information further. Nevertheless, the recurrent neural network architectures remain very complex, yet, hyper-parameters could be tuned for obtaining efficient functionality for IoT resolutions. This hypothesis puts motivation for applying DL perceptions to the security of IoT network.

1.3. Problem Statement

The relevance of security in contemporary connected world needs analyzing and processing a colossal amount of diverse data already; IoT network makes it even more complex. In spite of the several countermeasures proposed by many research studies, the IoT environment remains very prone to intruders. This clearly expose that gaps do

still exist on how IoT security concerns are mitigated. This work's central objective is to examine the use of deep-learning neural network in the detection of network intrusion attempts or attacks on a computer network in general and on an IoT network in particular. The neural network model to be used in the investigation is called Bi-directional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN). Also, this research stretches to inspect and provide an efficient way for implementing the deep learning approach and enhance its efficiency to detect intrusions in the IoT network infrastructure in competent and timely manner, attaining high degree of detection accuracy and low rates of false alarm.

1.4. Research Questions

The research questions of this endeavor are as follows:

RQ1. Why Deep Learning is efficient in intrusion detection accuracy over IoT network than prevailing machine learning techniques?

RQ2. How efficient is Bi-directional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN) in detecting intrusions in an IoT environment?

RQ3. What are the parameters essential for BLSTM RNN to generate a low False Alarm Rate (FAR) and a high detection accuracy?

RQ4: What are the efficient ways to implement the BLSTM RNN approach?

1.5. Research Goals

In answering the above-mentioned questions, the subsequent research goals would be achieved:

- Determine the key factors that give advantages to Deep Learning (DL) over prevailing Machine Learning (ML) techniques in detecting intrusion over IoT network.
- Implement BLSTM RNN approach using TensorFlow framework for developing the code for an AI model proficient of detecting intrusions in IoT network.

- Determine the probable optimal hyper-parameters required by the proposed model to attain the highest detection accuracy and FAR in least time.
- Evaluate the reported performance of the introduced model.

1.6. Delimitation

This thesis is limited to the detection of intrusions in IoT network layer only. Also, the proposed model is restricted to only detecting intrusions and provides no prevention mechanism whatsoever.

1.7. Research Contributions

This research work is a multidisciplinary venture that involves Artificial Intelligence (AI), IoT and computer network security. Therefore, a considerable amount of time is committed in interpreting the complexity of the perceptions in depth. We begin with recognizing the attack categories consisted in the intrusion recognition dataset. Then, learning the architecture of the IoT network and started assessing ML algorithms substantial for the IoT environment. However, we observed that deep learning algorithms forms the utmost appropriate methodology for the defined research problem. Experiments are performed using Google TensorFlow. The performance outcome are then evaluated and discussed. This inclusive interdisciplinary applied approach made the research work unique.

The new contributions of this research work are:

- Introducing BLSTM RNN for intrusion detection in IoT network.
- Design and develop the proposed BLSTM RNN algorithm using Python as programming language, and Tensorflow as implementation framework.
- Provide parameters tuning details that enables a low FAR and an elite intrusion detection accuracy.

1.8. Outline

The structure of this thesis is in the subsequent order: **Chapter 1** - Introduction: provides an introduction to IoT and introduces neurons and neural networks. Also gives an overview of the deep learning, activation functions and loss functions. This chapter also establishes the problem statement, research questions, research goals and research contributions along with the research delimitations. **Chapter 2** – Literature Review: discusses Internet-of-Things (IoT) along with IoT architecture and security concerns and the sectors where IoT systems are applied. This chapter also establishes the background in Machine Learning (ML), Deep Learning (DL) techniques and their implementation in IoT. This chapter also presents the literature review of similar works done by other researchers, existing methods and their issues and identify a potential research gap. **Chapter 3** – Neural Networks: discusses architecture and functionality of neurons, Feed-forward Neural Networks, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) RNN and Bi-directional LSTMs (BLSTMs) along with the neural network training procedure, activation function and loss functions. Also discusses about neural network implementation framework and UNSW-NB15 training dataset. **Chapter 4** – Research Methodology: provides the research methodology, data flow chart, implementation methodology (Keras and TensorFlow) and evaluation parameters. **Chapter 5** – Architecture & Implementation: proposes the architectural details and hyper-parameter values of the proposed model. Also discusses about the implementation of the aforementioned proposed model through Keras library and Python programming language. **Chapter 6** – Results: shows the simulation outcomes and the performance evaluation of the proposed model with various parameters. **Chapter 7** - Conclusions and Future Work: presents the conclusion drawn out of the project including the possible future works that might be accomplished to enhance and upgrade the project.

Chapter 2

LITERATURE REVIEW

2.1. Introduction

With progressively deep incorporation of human society with the Internet, the way people live, work and study is changing. Along with it, numerous security concerns are growing more serious. Identifying various network attacks, remains an inevitable technical concern. An Intrusion Detection System (IDS) could recognize attacks which are ongoing or an invasion that has already happened. As a matter-of-fact, the mechanism of detecting intrusion is equal to a classification task, including multiclass classification or binary classification. Precisely, the key motivation of detecting intrusions is to improve the classifier's detection accuracy in efficiently identifying abnormal data patterns.

2.2. Internet of Things

Internet of Things (IoT) could be realized as a persistent network of networks: plentiful heterogeneous things or entities, both virtual and physical like sensors, people, software and all types of devices) connected with some other entity or number of entities over unique addressing protocols and communicating with each other for providing services. The development of internet of things by utilizing the new IP address (IPv6) version, which drives beyond the confines of IPv4, will revolutionize the world of Internet by the connectivity for a huge number of connected smart devices nearby 70 billion, may be even more. Thriving this technology is called as the Second Economy or the Industrial Internet revolution [108].

2.3. IoT Definition

In the year 2005, the International Telecommunication Union (ITU) added to the conception of IoT and recommended 4 technologies to comprehend IoT: Intelligent embedded technology, RFID technology, sensor technology and nanotechnology [105]. As per [106] IoT is in yet in early phases and any regular or common standard to comprehend IoT hasn't been established however. There exist various definitions of IoT. For example, [104] explain IoT as environment of computing of several RFID

entrenched things which intercommunicates to provide smart Information Technology (IT) facilities.

The European Research Cluster on the Internet-of-Things (IERC) specifies IoT as: “a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual ‘things’ have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network” [107].

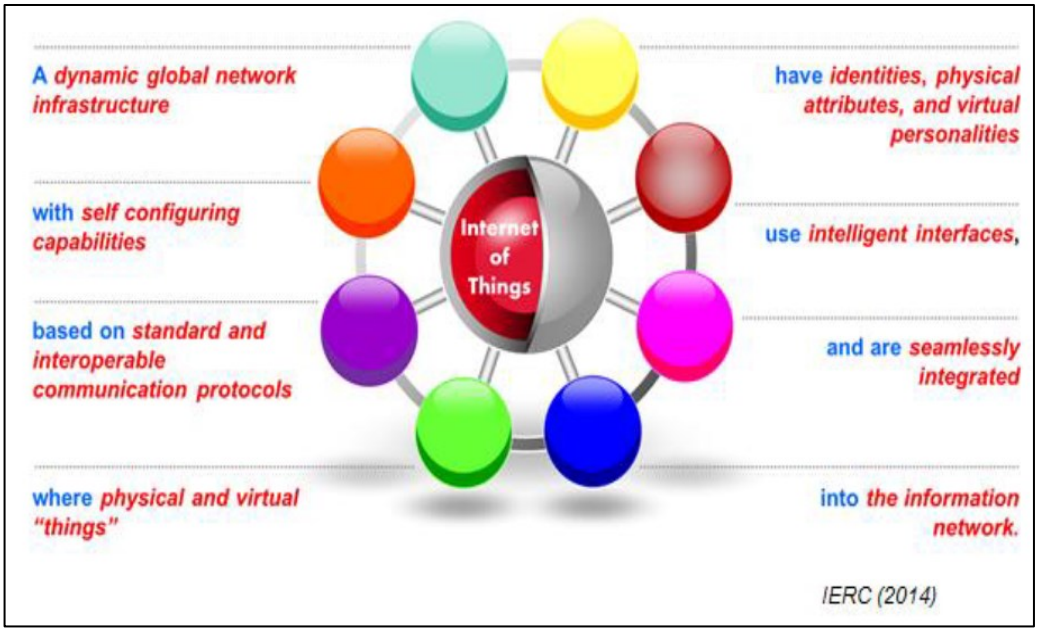


Figure 2.1: IERC definition of IoT [107]

Above figure illustrates IoT characterization proposed by IERC. “Dynamic global network infrastructure” refers to the network which is capable of automatically delivering and resource allotting in order to fulfill the prevailing and upcoming needs of IoT gadgets. This could be applied by using software defined networks (SDN) and cloud computing (CC) which allow “self-configuring capabilities” built over “standard communication protocols” like UDP and TCP/IP. Moreover, these protocols required to be implanted within the things that have “intelligent interfaces”. Intelligent interface possesses the capability to perform certain functions as per the need, such as connecting to the Internet, exchanging data and information etc. in a “seamlessly integrated” way like present personal computers.

2.4. Classification of “Things” in IoT

According to [105] the mapping among “things” over cyber and physical world is an inevitable part of IoT infrastructure, where “things” could be categorized into two sorts: Physical things and Cyber things. The grouping of “things” in IoT are illustrated in the Figure 2.2.

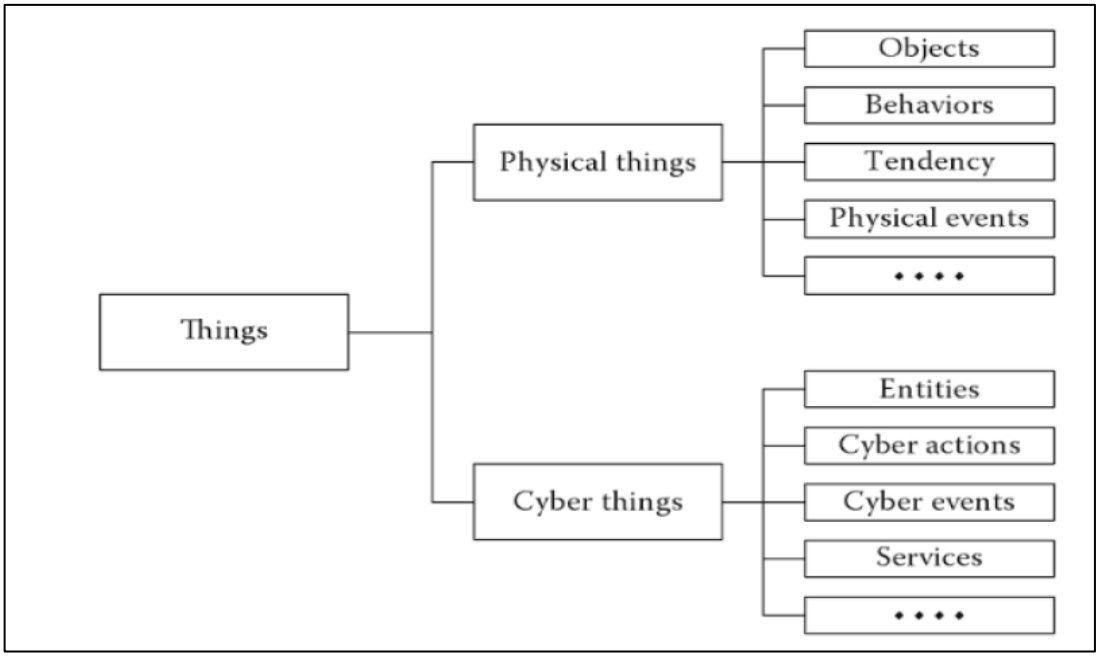


Figure 2.2: Classification of "things" [105]

2.4.1. Physical things

a) Objects: These are tangible things with measurable bodies like Persons, vehicles, tablets etc.
b) Behaviors: It refers to the movements of the objects. For instance, running, driving, monitoring and so on.
c) Tendency: This refers to the trends in physical things, like the tendency of a vehicle in a parking is to be stationary. This trend may also occur due to external factors like congested traffic or weather becoming cloudy.
d) Physical events: These are an assortment of all the above-named properties integrating to define the events caused by certain situations in the physical world.

2.4.2. Cyber Things

a) Entities: refers to the abstract things like code and data.
b) Actions: denotes data processing like, transmission of information with in entities.
c) Events: refers to entity activities like, reporting.
d) Services: refers to the tasks that are being offered to a thing or by a thing in order to perform a specific goal.

For instance, Google's automated car, exhibit a device which has comportments like driving, parking and so forth. The vehicle could have an affinity as when it discovers that it has begun to rain, it is likely to take physical activities like actuating wipers. The car wiper consists an algorithm which signifies the abstraction which process and share data dependent on available services for taking actions and reporting events.

2.5. IoT Architecture and Security Concerns

Despite the massive potential of the IoT in numerous spheres, the entire communication setup of the IoT network is flawed as per the security standpoint is concerned. The rising usage of IoT devices requires a prevailing security against probable vulnerabilities or attacks. Therefore, security is essential at every layers of IoT infrastructure, primarily for there is no network boundary or perimeter. Security constraints that required to be considered in IoT applications could be characterized into four key categories [109]:

a) Confidentiality: Confidentiality encompasses discretion in shielding data secrecy from third parties.
b) Integrity: For the sake of preserving the integrity of the information, the recipient of the message needs to authenticate that the received communications remained unaltered during the delivery or broadcast.
c) Authentication: Authentication in the IoT is the process of conforming that the communication actually is, from where it claims to be
d) Availability: It states the ability of accessing the information or any resources in the truthful format. An authenticated user could use several services of IoT to prevent Denial of Service (DoS) attack and keep the services available. DoS attacks remains a major threat to the availability [110].

Since IoT is the incorporation of multiple diverse networks, consequently it is challenging to accomplish a reliable association between the explicit nodes because of the constantly varying characteristics of the nodes. IoT architecture can be broadly arranged into three layers: sensing layer, transportation layer and application layer. Figure 2.3 below illustrates the security architecture of IoT.

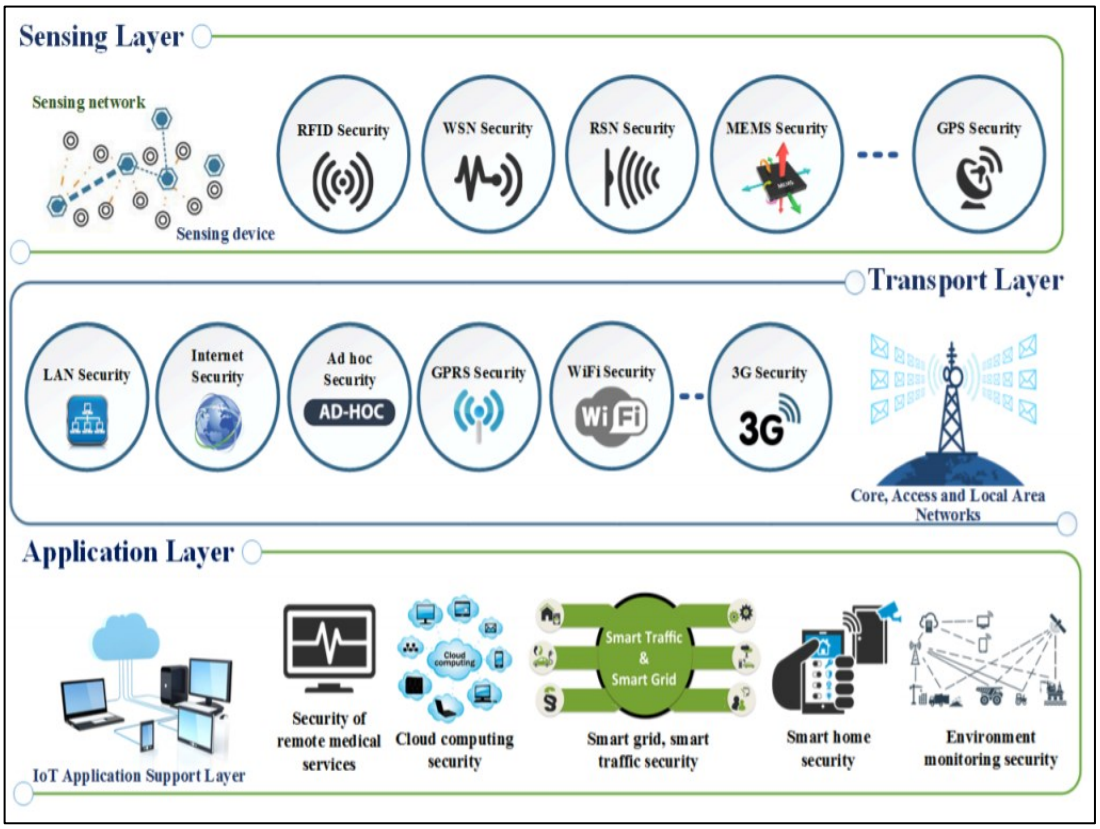


Figure 2.3: IoT security architecture [109]

Some of the most prominent security concerns are discussed below:

2.5.1. Perception Layer Security Issues

Perception or Sensing layer primarily consists of devices like RFID tag, Reader, Smart card, Sensor network etc. These devices remains exposed to subsequent vulnerability that may lead to security issues of IOT network like radio interference, sensor abnormalities and sensor attacks. [109]. Collecting real-time information needs a huge number of terminals are required at the sensing layer. This procedure requires data integrity and authentication. Since the nature of the communication is wireless, the primary problems occur in sensing or perception terminals comprise tampering, confidential information leakage, copying, terminal virus and other issues [110].

2.5.2. Security Issues at Sensor Network

Sensor nodes are accountable for information transmission, collaboration and integration. Since they are battery-operated with minimum security defense, they may encounter complex security matters like:

- Summoning Malicious Codes: Harmful programs like worm could affect the sensor network very easily, as, to execute, the worms do not require any other dependent files, which makes it very hard to identify and take action.
- Tag defect: Due to lack of enough security it is easy for the intruder to accomplish illegal usage of legal reader. An invader could effortlessly get the tag information and for accessing Radio Frequency Identification (RFID) devices devoid of any kind of prior authentication through forging.

2.5.3. Security Issues at Network Layer

IoT network layer chiefly consists of Computers and Wired or wireless networks. The reason for this layer is to transmit data. Nodes move freely in wireless networks, i.e. nodes are able to connect or disconnect from the network anytime with no prior conformation which make wireless networks exposed to security threats. The network layer of IoT ought to have that capability to cope with such malicious obliteration. Nonetheless, mechanisms that exist is not adequate to deal with this security issue [109].

2.5.4. Application Layer Security Issues

IoT application layer comprises of diverse applications such as industrial monitoring, monitoring services, smart grid or any additional intelligent systems. Malicious program or software exposures could be hosted in such exposed systems. Additionally, the incorporation of various techniques and professional requirements could generate a congestion or bottleneck for processing immense data and on process control which might lead to the safety and reliability issues of IoT systems [110].

2.5.5. Network Capacity Limitation

The converging of devices which ascends from the IoT system kindles greater claim for a certain grade of Quality of Service (QoS) of the connected network infrastructure. Applications that deliver certain services might demand additional frequent transfer of small data blocks (sessions) essential for upgrading and synchronization. Frequency of the mentioned sessions will generate a significant effect on delay and penetrability of the network. This fragment of the infrastructure necessarily be securely brought for ensuring secure information flow [111].

2.6. IoT Challenges

IoT is yet at an early stage of development and encounters numerous challenges. Primarily, there exists no standard structural design for IoT networks [11]. Due to its initial phase, companies are not eager to manufacture devices which comply with other merchant's devices to accomplish monetary benefit and drive customers toward the sellers' lock-in. Besides, IoT networks are heterogeneous, as a result associating, operating, organizing and securing the network is a difficult task. Thirdly, IoT devices utilize diverse communication conventions to correspondence over various kinds of networks (e.g. Bluetooth, WAN, GSM, and WSN) [11]. Then comes privacy and security. Due to different types, and the amount of IoT devices, and their restricted equipment capacities, it is relatively difficult to employ host-based protection methods for securing IoT objects, which drives to the idea of network based security measures along with intrusion detection and protection techniques.

2.7. Machine Learning

Machine learning (ML) is considered as a sub area inside AI which is becoming ever more prevalent and is broadly employed in the industry and academia for solving various tasks. However, AI isn't new to computer science. It started back in the year 1950, after Alan Turing introduced famous interrogation of "Can machines think?"[37]. Later, the focus of AI has been distributed around various expanses. With the accessibility of enormous capacities of data, data-driven method of ML has become so prevalent. Mitchell [38] states ML by: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"[38]. Taking

the classical example of spam filter would put it into the proper context, which is: The task 'T' represents the prediction of whether an email is a spam or not. The 'E' represents the experience, that is the training data set, and the performance 'P' is measured as the ratio between appropriately classified emails.

2.7.1. Types of Learning in ML

According to [112] "learning" in ML can be categorized in to the following:

a) **Supervised Learning:** It is a systematization of the concept of learning from examples. The learner (commonly, a computer program) is provided with two separate data sets. One is called the training-set while test-set is the other one. Through the training-set the learner can "learn" the patterns of labelled instances and can recognize unlabeled samples in the test dataset with the highest possible accuracy [112].

b) **Unsupervised Learning:** Here, the data is clustered into classes depending on the statistical propertied if the input data. Unlike supervised learning, there remains no explicit target outputs related to each input. In unsupervised learning the learner receives inputs *x1, x2… n* and determines which aspects of the input structure should be considered in the output [112].

c) **Reinforcement Learning:** In this sort of learning both marked and unlabeled information could be utilized to form the fundamental knowledge. The framework gets a reward for every right or wrong forecast. Depending on the reward the next forecast could be generated. At the point when new data is given to the framework, the framework will endeavor to locate the best execution way or join in multiple execution pathway for forecasting and pause for the reward. When the obtained reward occurs to be superior with respect to the past rewards for the identical input, at that point, this pathway turn out to be agreeable Reinforcement learning is utilized in web based games, for example, Chess [112].

2.7.2. Algorithms of Machine Learning

Algorithms could be defined as an arrangement of rules to be carried out in an explicit order for resolving a specified problem [112]. Below are the list of groups of ML algorithms:

1) Regression Algorithm

It is a technique based on predictive modelling. It examines the correlation in between the dependent variable (also called as target) and one or multiple independent variables (also called predictor) [121]. Regression algorithms are commonly utilized for modelling sequential or time series data. Instances of usage are: forecasting, financial modelling including discovery of relations among variables. For instance, forecasting of stock prices are best performed using regression. It deals with the statistical data analysis and involves indicating significant relationships among a dependent variable **Y** and a set of independent variables X_1, X_2,…, X_n. Some of the most widely used examples are given below:

a) Stepwise regression
b) Logistic regression
c) Ordinary lest squares regression (OLSR)
d) Linear regression

2) Clustering Algorithm

Clustering algorithms are generally apprehensive with recognizing the input data patterns and establish them into groups or clusters. The goal is to place similar objects (as per specific similarity measure) in a same cluster and allocate unlike data to different clusters [121]. Data are generally defined and clustered by utilizing a set of values and features. It is an unsupervised form of learning since there exist no pre-existing data categorization. Some of the widely used clustering algorithms are as follows:

a) Expectation Maximization (EM)
b) K-Medians
c) Hierarchical clustering
d) K-Means

3) Bayesian Algorithm

Bayesian algorithm employ Bayes' hypothesis in regression and classification problems [121]. It considers the conditional probability of every conceivable reason for a certain outcome. Most widely used Bayesian algorithms are as follows:

a) Bayesian Network
b) Gaussian Naïve Bayes
c) Naïve Bayes
d) Multinomial Naïve Bayes

4) Decision Tree Algorithm

Decision tree is a competent nonparametric technique that could be employed either to regression or to classification tasks [121]. Most widely used decision tree algorithms are as follows:

a) Decision Stump
b) Classification and Regression Tree (CART)
c) M5
d) C4.5 and C5.0

5) Artificial Neural Network Algorithm

Artificial Neural Network (ANN) algorithms mimics the way of the biological neurons for complying with classification and regression problems [121]. Some ANN algorithm examples are as follows:

a) Radial Basis Function Network
b) Perceptron
c) Hopfield Network
d) Back-Propagation

There are additional ML algorithms, like association rule learning, deep learning, regularization and features selection algorithms, and dimensionality reduction – just to name a few examples.

2.8. Machine Learning versus Deep Learning

Machine learning (ML) is extensively utilized in identifying several types of attacks. A ML methodology could assist the network admin to perform required actions for averting intrusions. Nevertheless, most customary ML methods reside within shallow learning and usually give emphasis to feature engineering and selection. With enormous intrusion data that rises in the real-time network environment, shallow learning is unable to resolve the classification issue efficiently [29]. Deep learning approach possess the potential to extract improved representations from the dynamic data-sets, and is capable of creating much efficient prototypes. G. Hinton *et al*. [30] introduced the hypothesis of DL in the year 2006, and over years the theory underwent spectacular rise in the area of ML. Most recent ML methodologies work well because of human-designed representations and inputs features.

In ML, 'representation learning' or 'feature learning' represents set of methods that makes a system capable of automatically learning the representations required for detecting features from the training dataset. DL, on contrary, could be considered as establishing both representation learning and machine learning together. DL pursuits to together learn essential features along multiple levels of cumulative intricacy and abstraction and the concluding prediction. Figure 2.4 illustrates the fundamental difference between ML and DL, where traditional ML involves manual feature selection and on contrary DL employs automated feature selection.

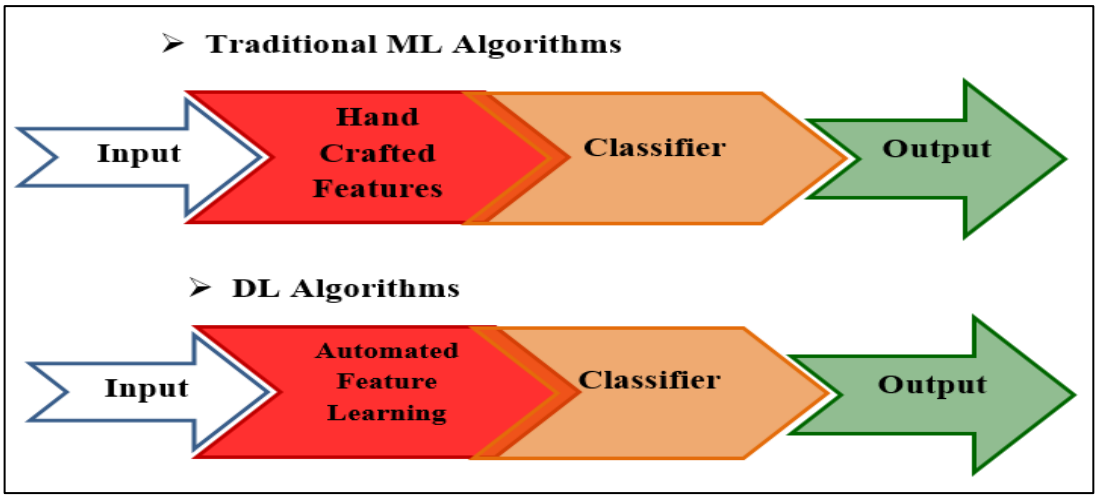


Figure 2.4: ML and DL

2.9. A Popular ML Algorithm - Deep Learning

Machine learning (ML) is comprised of several approaches, Deep learning (DL) is one of them. The basic principle behind every deep learning technique is the automated discovery of abstraction. DL comprises of supervised and unsupervised learning methods established on multiple layers of ANN. DL is comprised of manifold processing layers, where every layer generates a non-linear output from the input layer data. The DL functionality is inspired from the signal processing capability of neurons of the human brain.

Compared to other traditional ML approaches, DL models has gained more consideration in current years. Figure 2.5 demonstrates the popularity flow of 5 popular ML algorithms as per Google trends, where deep learning is fetching further popularity amid others. The emerging trend for Deep Neural Networks (DNNs) began since 2006 after G. Hinton *et al.* [42] proposed deep belief networks hypothesis. Subsequently, this expertise is employed in diverse areas of artificial intelligence including search engines, image recognition, natural language processing so on and so forth.

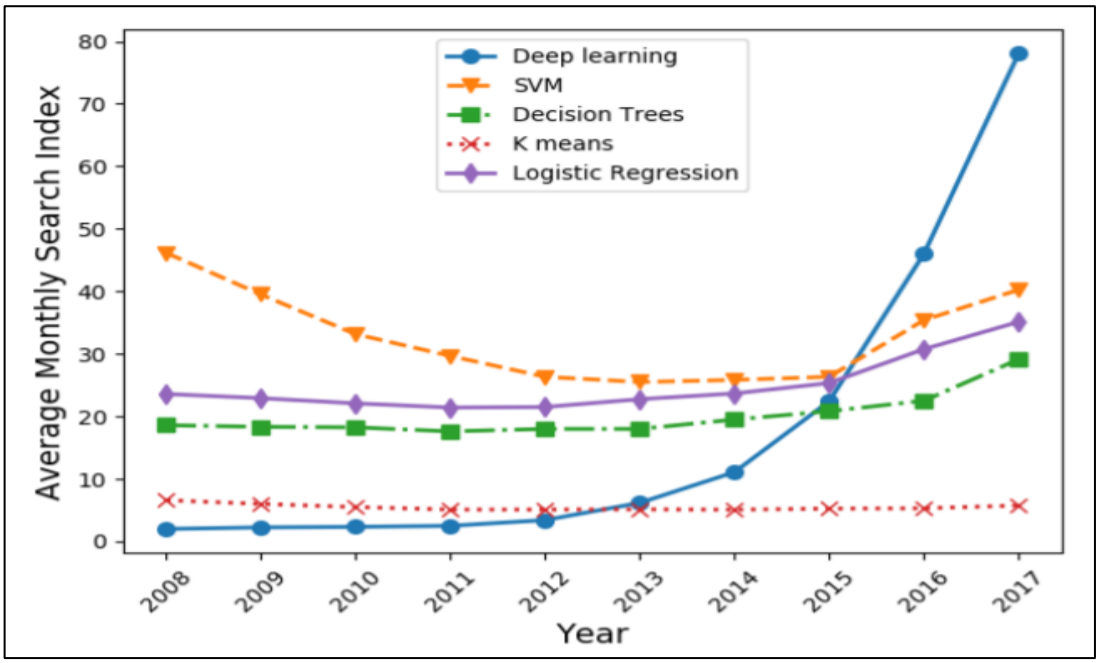


Figure 2.5: Google Trend screening more inclination toward DL in recent times [39]

2.9.1. DL Architecture

Deep neural network (DNN) comprises of 3 major layers, namely: the input layer, manifold hidden layers and the output layer. The layers are constituted with multiple neurons or units. A single neuron is the computational unit which accepts some input vectors, computes a weighted summation of the input vectors, then passes the resultant sum through the activation function for generating the output. Figure 2.6 represents the structure of a single neuron, where $\{X_1, X_2 \ldots X_n\}$ represents the set of inputs, $\{W_1, W_2 \ldots W_n\}$ represents the weight vector and the bias is represented by b. These weights and biases would be optimized through the training course. The summation of all the inputs, their respective weights and bias are feed into the activation function to generate the output. The purpose of the activation function is to help the neuron to learn complex patterns and present a non-linear properties into the network.

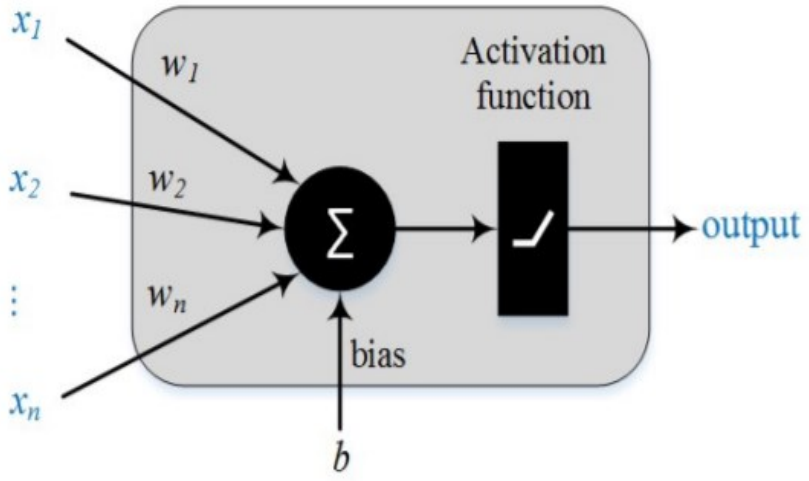


Figure 2.6: A neuron with multiple inputs and weights and bias [39]

In a typical DL input layer, random weights are allotted to the input data and are forwarded to subsequent layer. Every succeeding layer similarly allots weights to the respective inputs and generate outputs. Output of former layer contributes as input of the subsequent layer. Model's output layer represents the prediction outcome. The accuracy of the model is determined by a loss function that computes the error-rate among the actual output (i.e. output generated by the model) and the expected output. The loss or error-rate represents the divergence among the actual and expected output. The error-rate is then transmitted over the network back to the input layer. This

technique of error-rate transmission across the network is termed as Back-propagation (BP). The BP is utilized for updating network weights and biases. The DNN again iterates the cycle and optimizes the weights of individual neuron in every iteration, till the error-rate reduces under an anticipated threshold value. Once it's attained, the DNN is trained and is equipped for operation. Figure 2.7 illustrates the high level working of training phase of a typical DL algorithm.

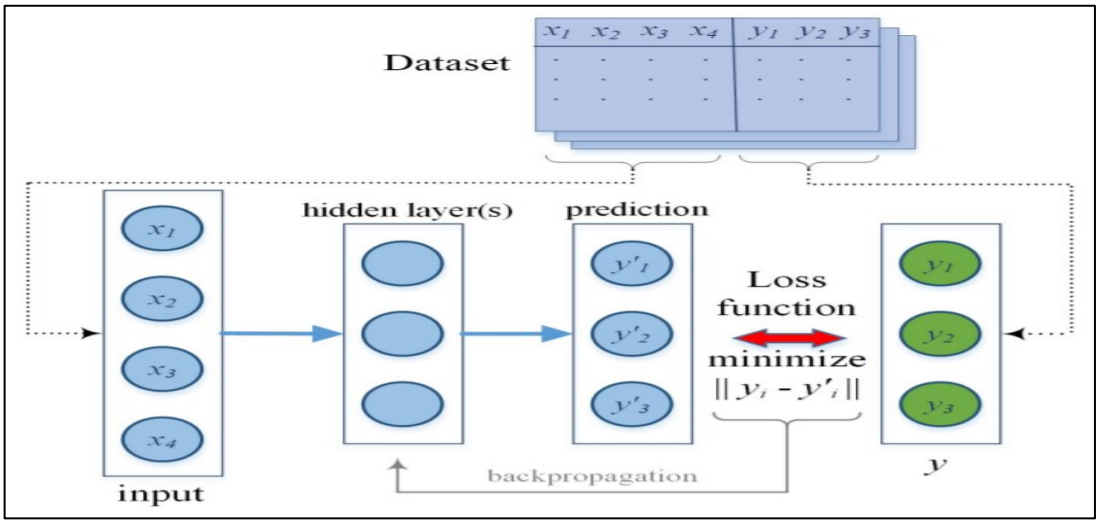


Figure 2.7: Overall working of DL training [39]

2.9.2. Salient Aspects of Deep Learning

Four main reasons of deep learning resurgence are discussed in the following sections.

2.9.2.1. Representation Learning

Representation learning or feature learning is a mechanism which makes a system capable of automatically determining the representations required for classification or feature recognition [6]. Handcrafting features is excessively time-consuming, and features are frequently both incomplete and over-quantified. Additionally, effort must be given again for every modality task like text, images, databases or even language and domain. In contrary, if ML could learn features automatically, then the entire process of learning possibly will be automated more simply, and various additional tasks could be resolved. DL delivers one means of automated feature learning which substitute hand-crafted feature engineering by

making a machine capable of both feature learning and using those features for performing an explicit task.

2.9.2.2. Distributed Representations

This signifies many-to-many association amongst two kinds of depictions. For example, patterns and neurons, where an individual pattern is represented by several neurons and individual neurons partakes in the representation of several patterns [7]. Many shallow learning models faces the problem of so-called "curse of dimensionality." Since an index vector over a huge data volume is very sparse, hence, the models can simply overfit to the training data. The traditional solutions to this kind of issue encompass either hand crafted feature engineering or the procedure of very simple target functions like linear models. Deep learning technique generally use distributed vector representation as an alternative to discrete vector counts which makes the models more robust. DL network could learn in an unsupervised way to apprehend distributional resemblances and also can be fine-tuned in a supervised manner.

2.9.2.3. Learning Multiple Levels of Representations

Deep learning algorithms are particular cases of representation learning with the feature that they learn multiple levels of representation. For instance, deep learning architectures like convolutional neural networks (CNN) [31] trained on images are capable of learning similar levels of representations as human brain does. The 1st layer acquires knowledge on simple edges, the 2nd layer learns primeval shapes and the higher layer combines all these to produce objects.

2.9.3. Resent Advances

Neural networks (NNs) are around for several decades [32]. Nevertheless, till 2006, deep NNs were overtaken by shallow architectures. In the same year, though, Hinton and Salakhutdinov [33] proposed a unique technique of pre-training the DNNs. The concept was based on employing restricted Boltzmann machines for initializing the weights of a single layer at a time. This acquisitive technique initialized the weights of the fully connected NN which resulted to enhanced local optima [34]. Vincent *et al.*

[35] exhibited that alike effects could be achieved by utilizing auto-encoders. An auto-encoder is an ANN employed for unsupervised learning.

Additional causes have lately facilitated deep learning networks to attain state-of-the-art performance. For instance, accessibility of big datasets, faster computing devices like multi-core CPU and GPU computing architectures. A deep learning architecture excludes manual feature engineered training data and hence requires an enormous size of data. In this era of 'big data', various institutions and researchers can inexpensively and easily accumulate huge datasets that might be utilized for training DL models with many parameters.

2.10. Implementation of DL in IoT Applications

2.10.1. Smart Homes

Smart homes incorporates a broad scope of IoT applications which could upgrade homes' energy utilization and the quality of living of their occupants by giving intelligent services. For instance, for collecting information from fridge interior, Microsoft and Liebherr are implementing Cortana DL [56]. These analytics and forecasts could profit the home to have a smart and efficient electrical power system, and with additional peripheral information, could be utilized for forecasting and checking health orientation. Due to continually increasing demand of household electricity, the capability to regulate and enhance energy competence and forecasting the forthcoming need is becoming a requirement for smart homes. Other instances like, forecasting electricity load in a smart home forms the basic applications that employs diverse DL algorithms [57]. Manic *et al.* [57] did an energy load analysis for home energy ingesting by employing different DL models, namely, CNN, LSTM and LSTM Sequence-to-Sequence (S2S) and showed that LSTM S2S performs a better forecasting than DL models.

2.10.2. Smart City

Smart city incorporates a few other IoT areas, to be specific, transportation, agriculture, energy and so forth. Nevertheless, smart city deals with heterogeneous data that gets generated from diverse areas and leads toward big data. By means of deep learning architecture, analysis of big data could yield high quality performance [39]. In order to see the usefulness of DL models over IoT environments, Toshiba

conjointly with Dell Technologies has lately developed a DL testbed, and used it in a Smart Community Center, Kawasaki, Japan, for evaluating and analyzing the gathered information [91]. The big data which fuels the testbed were collected from construction management, building security and air conditioning. Another significant matter for smart city is prediction of patterns of the crowd movement. Song *et al.* [58] established a mechanism built upon DNNs for resolving the issue on a city level. Their proposed model is based on 4-layered LSTM RNN employed to learn from human mobility (GPS information), joined with modes of transportation like train, car, bicycle, and walk). The authors insist that their approach of deep LSTM RNN attains better efficiency than shallow LSTMs. Waste supervision and classification of trash is one more correlated job that smart cities should exhibit. A vision-based classifications by utilizing deep CNNs might be a way to address the job [59]. Amato *et al.* [60] established a decentralized structure for identifying the occupied and the vacant spots in a parking lots by means of smart cameras and deep CNNs. Valipour *et al.* [61] also came up with a detection system for vacant parking areas by employing CNN and exhibits better results than SVM network.

2.10.3. Energy

Smart grid refers to a power supply network that utilizes digital communications technology for detecting and reacting accordingly to the local variations in utilization. Predicting the energy sources like wind, solar and other natural resources is evolving as a dynamic research area. DLs are progressively employed in various applications of smart grid. For instance, Gensler *et al.* [62] inspects the effect of various DL architectures and their evaluation shows that the combination of LSTMs (Auto-LSTM) and AEs (Auto Encoder) and generate the best outcome compared to other methods of learning. Muranushi *et al.* [63], proposes a web-based LSTM RNN forecasting system for prediction of the solar power.

2.10.4. Intelligent Transportation System

The Intelligent Transportation System or (ITS) refers to the technology of detecting, investigation, control and intercommunication advancements to build transportation in order to enhance security, portability and productivity. Ma *et al.* [64] proposed a system built on DL architecture that analyses the transportation network by utilizing the GPS data as the model input. The proposed model generates a high

accuracy of 88%. Y Tian *et al.* [65] also conveyed the study on short-term traffic flow forecast by means of LSTM RNN model that showed enhanced prediction accuracy in comparison with other models like support vector machine (SVM), stacked Auto Encoders (AEs) and traditional feed forward NN. In [66], ITS data are fed to an IDS built using DNN in order to facilitate in-vehicular network communications security. Moreover, it inspires the progress of methodologies used for traffic signs recognition. For example, technologies like autonomous driving, mobile mapping and driver assistance systems require such mechanisms for providing consistent services. Cires *et al.* [67] introduced a DNN based system of traffic sign recognition and stated increased accuracy with the methodology. Additionally, self-driving vehicles utilize DNNs to execute various jobs, like detecting pedestrians, obstacles, traffic signs etc.

2.10.5. Healthcare and Wellbeing

Internet-of-things in collaboration with deep learning is utilized for serving healthcare solutions and prosperity for societies and individuals. For instance, Liu *et al.* [68] introduced a CNN based image recognition system for recognizing images of food and their pertinent facts (like portion and types). In healthcare, Pereira *et al.* [69] utilized the concept of handwritten image identification by employing CNNs which helps detecting Parkinson's disease in its initial stages. In [70], DL have been used to recognize cardiovascular diseases from mammograms. The study established a twelve-layer CNN for identifying the presence of breast arterial calcification (BAC). Their outcome shows that the precision of the proposed model is parallel to human experts. Lipton *et al.* [71] studied the performance efficiency of LSTM RNN network for analyzing and recognizing multivariate time sequence patterns of medical extents in intensive care units (ICUs). A study of DL in the field of health informatics is provided in [72]. Researchers have also utilized time sequence medical information for forecasting and diagnosis of diseases through RNN based architecture.

2.10.6. Agriculture

Disease identification in plants by means of DNN models proved to be an effective measures. Sladojevic *et al.* [73], proposed a disease identification system for plants which is built on the cataloging of the leave pictures by using CNN model coded by the Caffe framework. Such identification model could be employed as a mobile app for the agriculturalists to recognize plant disease by clicking leave images. DL has also

been applied in remote sensing for crop and land recognition and gradation [74] [75] [76]. Another instance, DL has been used for predicting and detecting in the area of automatic farming. Steen *et al.* [77] introduced a DL based model of deep CNNs to detect obstacles in the agricultural land. This approach helps the autonomous machineries to operate safe and sound over the field. Furthermore, in automated harvesting, detecting the stage of fruit (ripe or raw) is crucial. Sa *et al.* [78] employed a variant of deep CNNs called Region-based CNN for studying the fruit images.

2.10.7. Education

IoT combined with DL are capable of contributing to the effectiveness of the current education system. Enhanced reality technology united with mobile devices forms potential implicating area for DL techniques. The combined technologies of IoT and DL would help to keep students encouraged, studies and trainings to be stimulating, and making means of learning to be competent [39]. Additionally, DL could be utilized as a customized recommendation unit [79] to endorse more pertinent contents to the instructor. The utilization of DL in supplementary areas, like summarization of text and translation of natural language, will be beneficial for smart education. Yang *et al.* [80] used a technique for grade prediction of students in MOOCs (Massive Open Online Courses). The authors have utilized the clickstream data gathered from lecture videos while students were interacting with the videos while watching. Those clickstream data were then fed into a DNN model which learns from both clickstream data and former performance. Moreover, Piech *et al.* [81] used LSTM and RNN architecture for predicting educators' answers to quizzes and exercises, based on their former actions and communications over MOOCs. Monitoring classroom occupancy is one more application studied by Conti *et al.* [82]. In their investigation, the authors introduced twofold approaches intended for estimation of density estimation and head detection. Both the proposed approaches are constructed on CNN for calculating the student number in a specific classroom.

2.10.8. Industry

In industry, cyber-physical systems (CPS) and IoT forms the central essentials to advance manufacturing technologies delivering high-accuracy and intelligent systems [39]. Luckow *et al.* [83] investigates a visual inspection by using CNN network with AlexNet and GoogLeNet. In this study, various images of vehicles along with their explanations are fed to a DL model. The system uses TensorFlow framework and shows that the best efficiency acquired is accuracy of 94%. Shao *et al.* [120] employed DNNs in a fault identification system aimed for extracting important features by utilizing denoising auto encoder (DAE) and contractive auto encoder (CAE) and. In another study, Lee [46] proposed a model in combination with IoT and cloud platform for sustenance of error recognition of defect categories in car headlights in automobile manufacturing and the outcome established the better efficiency of the DBN model over SVM and RBF (Radial Basic Function). In [11], the authors has proposed stacked denoising auto-encoders (SdA) for two purposes: one, sensory data noise reduction that happened due to electrical and mechanical turbulences. Second, for performing classification of faults. They experimented their proposed approach in wafer samples of a photolithography process and the reported outcome revealed that in noisy situation, the proposed system generates 14% higher accuracy with respect to other methods including SVM and K-Nearest Neighbors.

2.10.9. Government

Governments could fetch inordinate potential benefits by utilizing intelligent and enhanced connectivity that originates from the merging IoT with DL. For occurrence, the prediction and recognition of natural disasters like forest fire, landslide etc. and ecofriendly monitoring is of high priority. Optical remote sensing imageries were provided as input into a deep AEs model along with softmax classifiers to forecast environmental landslides were introduced by Liu et al. [124] in 2016 with a reported accuracy of 97.4%, consequently beating SVM and ANN models. Another investigation done in [84], employs LSTM RNN for the earthquake prediction by using the US Geological Survey website data for the training purpose. Their experiment reported 63% accuracy with 1-Dimentional input and 74% accuracy with 2-Dimentional input. Also, Liu *et al.* [85] introduced a model based on CNN for extreme climate events recognition (like weather fronts, atmospheric rivers and tropical cyclones). They fed the system with picture patterns of weather events. The system is

implemented in Neon framework with an accuracy of 89%- 99%. Additionally, [86] addresses the issue of road damage detection by utilizing DNN architectures which gathers its data from crowd-sourcing empowered by IoT devices. The study is performed through a deep CNN and evaluations shows a damage identification accuracy of 81.4%.

2.10.10. Sport and Entertainment

Sports analytics are drastically evolving. However, DL is very new in this sector and very few studies have been conducted using DNNs. In [87], a DL approach is proposed for creating an intelligent basketball ground. The system uses SVM for choosing the finest camera for real-time propagation and provide basketball energy images to a CNN network henceforth delivering correct online score and fascinating highlight clips with an accuracy of 94.59%. In another study [89], Wang *et al.* presents an RNN for doing grouping of invasive basketball plays over NBA games which uses SportVU3 dataset with 80% accuracy. Kautz *et al.* [90] examined players' action identification in volleyball. A CNN architecture along with wearable sensor data were used in this study and achieved classification accuracy of 83.2%. Group activity identification forms another exciting course for sport teams. In [91] Ibrahim *et al.* studied this by employing hierarchical model based on LSTM RNN in volley ball team. In their study, to derive the events for every player they constructed a single LSTM network, along with a top-level LSTM network to sum-up the discrete models for overall team conduct identification. In comparison with the other models, introduced hierarchical architecture attained enhanced outcomes.

2.11. Related Works

A recent work by B. A. Tama and K. H. Rhee [4] proposes a DNN methodology where instead of employing outdated datasets, like NSL-KDD and KDDCup 99, the authors have evaluated the DNN performance over three contemporary IoT related benchmark datasets, namely: GPRS, CIDDS-001 and UNSW-NB15. The accuracy measure of the model is tabulated in Table 2.1. The study also reports an occurrence of bias results in CIDDS-001 dataset due to data imbalance issue, which is the distribution of one class in CDDS-001 dataset is compellingly lower than the supplementary class [4]. The study also remains unable to observe the performance differences between DNN and other algorithms. In this study, UNSW-NB15

benchmark dataset is chosen for evaluating our proposed BLSTM RNN model for detecting intrusions in the IoT network.

Table 2.1: Corresponding dataset and reported accuracy

Dataset	**Accuracy**
UNSW-NB15	94.04%
CIDDS-001	99.99%
GPRS-WEP	82.74%
GPRS-WPA2	92.48%

In recent years, deep learning has developed progressively, and has become functional for detecting intrusions and outperforming conventional methods. Studies reveals that DL entirely outperforms conventional shallow learning methods. In [12], a deep learning method has been used by employing a DNN for flow-based anomaly recognition. The outcome reveals that the proposed technique could be used for detecting anomalies in software-defined systems. In [13], a deep learning technique has been proposed where the authors use a self-taught-learning (STL) algorithm over NSL-KDD dataset. When relating the performance with former studies, the approach has proved to be more efficient. However, their studies emphasize only on the feature reduction capability of DL techniques. Fu *et al.* [5] introduces a novel technique for intrusion detection intended for the IoT systems established upon anomaly extraction. In their study, the authors assert that anomalies are detectable by analyzing the patterns of the data of the IoT sensor layer, like the temperature, humidity or anything that an IoT object sensor could collect and report. The study uses an unsupervised algorithm for data-mining for identifying normal patterns. For performance evaluation, *Intel Lab Project* dataset was employed, but no detected accuracy was reported to the designed system. Another study conducted by M. Sheikhan *et al.* [20] claims that RNNs can be viewed as reduced-sized neural networks (NNs). The paper introduces a 3-layer RNN architecture having 41 input features and 4 intrusion classes as outputs for a misuse-based intrusion detection system. Nevertheless, the RNN units of layers remain partly connected. As a result, the proposed RNNs does not exhibit the capability of DL to

produce high dimensional features. Moreover, performance evaluation of the proposed approach in terms of binary classification has not been reported.

With the consistent growth of big data along with the increase in computational power, the deep learning technique has become popular rapidly, and is increasingly utilized in numerous fields. This thesis work introduces a unique DL to detect intrusions over IoT network by employing a bidirectional LSTM (BLSTM) recurrent neural network (RNN). Related with former works, we have used the BLSTM-based model aimed at binary classification and excluding pre-training. In addition, we have used two distinct data sets for training and testing purposes (namely, *UNSW-NB15_training-set.csv* and *UNSW-NB15_test-set.csv*) for evaluating the performance of the proposed model.

2.12. Improvement to Existing Identified Research Gaps

There exist several research gaps within the prior related works. Foremost, no studies has been conducted using both BLSTM RNN and TensorFlow implementation framework in order to detect intrusion in the IoT network. Second, most of the previous work has used the traditional RNN that has the exploding and vanishing gradient [15] problem, which gets resolved by LSTM RNN. But, LSTM network has a major limitation, that, it cannot be trained in both positive and negative time direction [28]. As a result, during training phase, the LSTM network needs to search for "optimal-delay" (another extra parameter needed for training) of the network. Eventually, while the delay becomes so big that nearly none of the vital data could be saved, then the NN congregates to the probable optimal resolution depending on the prior information [28]. Bidirectional LSTM (BLSTM) RNN resolves the problem of optimal delay, since the BLSTM architecture propagates the existing data in both forward and backward direction in time [28]. We have attended this research gap in our work by proposing a novel Bidirectional LSTM RNN architecture for intrusion detection. Third, most of the prior works used benchmark dataset like KDD'99, NSL-KDD etc. which remain highly criticized. In [20] the authors' express that the KDD dataset is obsolete and endures with data redundancy data which may prompt to partial detection accuracy. In [27] the authors insist that the NSL-KDD dataset comprises of redundant occurrences and it is not appropriate to be used for the accurate training of NN models. Fourth, a very limited amount of work has been done to detect intrusion in the IoT network using deep learning technique. This piece of work contributes to the literature of IoT network intrusion detection mechanism. This research work uses UNSW-NB15 dataset, which

according to the literature forms to be the most recent and effective dataset published for intrusion detection research work purpose.

Chapter 3

NEURAL NETWORKS

3.1. Artificial Neurons

An artificial neural network (ANN) is a widely used ML architecture inspired by the functioning and structure of human cerebrum. Any neural network consists of basic computational components termed as neurons. Essentially, neurons take input vectors, multiplies the input vectors with their respective weights, then sum up all the multiplications, and then employs a non-linear mathematical function termed as activation function, which is to compute neuron output. Figure 3.1 illustrates the operation of a single neuron.

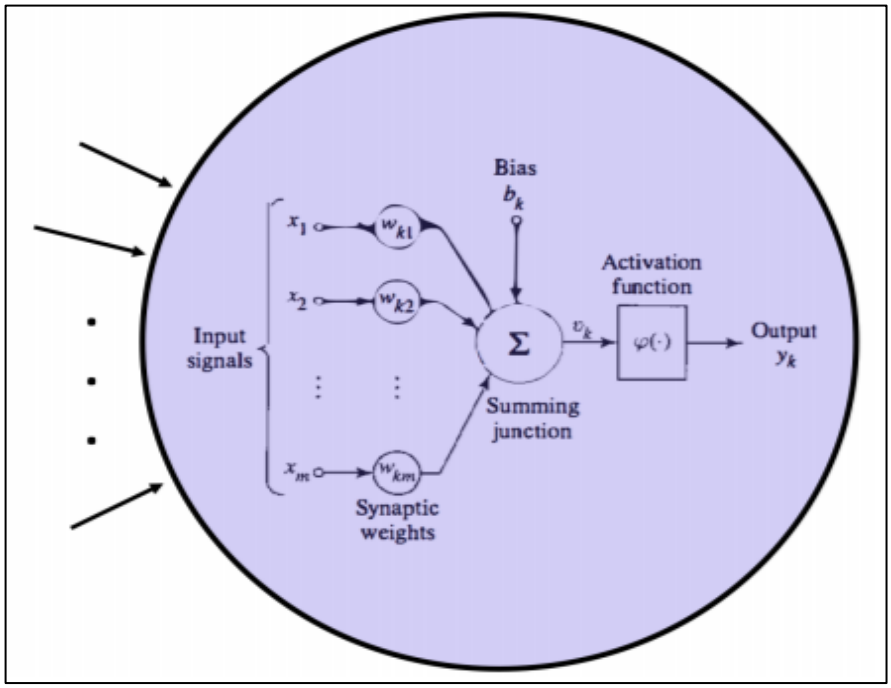


Figure 3.1: Working of a single neuron [113]

A neuron output can be calculated mathematically by (1.1), where, Y_k denotes output of the neuron, *f()* signifies the activation function, W represents weight of each input(s), $\otimes$ represents element-wise multiplication, X represent input vector and b stands for the neuron bias. X_i represents the input vectors where, i = 1, 2, ...,, n. The input signals are multiplied by corresponding weight W_{ki} where k represents the neuron number, i represents the input signal number. V_k represents the net input and is

calculated by summing up all the input vectors. Moreover, another input called bias (b_k) is also feed into the network. The computation of the net input vector V_k is shown in (3.1).

$$U_k = \sum_{i=0}^{n} W_{ki} X_i \tag{3.1}$$

Where, $X_0 = 1$ and $W_{k0} = b_k$. The output of the neuron Y_k is calculated by (3.2). To perform the computation an activation function $\varphi(\cdot)$ is employed on the net input V_k:

$$Y_k = \varphi(V_k) \tag{3.2}$$

3.2. Feed Forward Neural Networks

Feed forward neural network (FFNN) is a multi-layered structure made up of many neurons or units. A neuron of a FFNN remains fully connected to each other. For instance, in Figure 3.2, each and every neurons of the input layer remains connected to each and every neurons of the hidden layer. The connections between neurons are referred to as edges. Every edge or connection is associated with respective weights. The foremost FFNN layer is termed as the input layer, as input to the network is received through this layer. The last layer of the FFNN is termed as the output layer since it delivers the network output. Rest of the network layers are conjointly stated as hidden layers. FFNN are useful for performing supervised learning tasks [5]. Figure 3.2 shows a FFNN architecture where each and every neuron remains connected with all other subsequent layer neurons. This type of architecture is termed as fully connected NN.

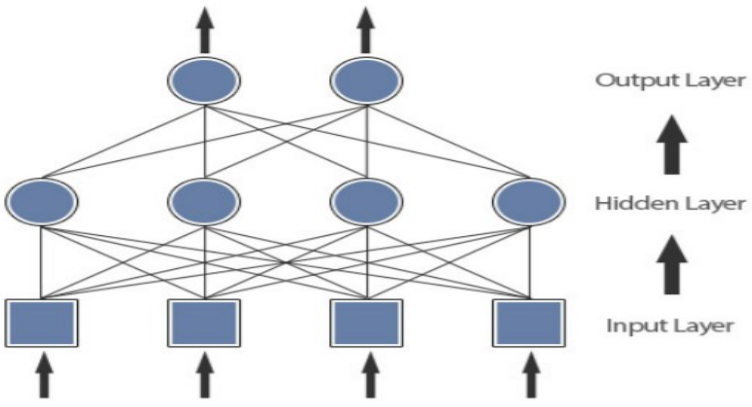


Figure 3.2: Feed-forward Neural Network [115]

3.3. Recurrent Neural Networks

Recurrent Neural Network (RNN) is comprised of several layers with feedback loops and is able to propagate past information onward to the present time. An RNN consists of loops and these loops allow the information to persist. The hidden layers of the RNN act as information storage like computer memory. RNNs form a class of powerful DNNs that use its internal memory along with loops for dealing with sequence data [47]. Figure 3.3 illustrates an RNN neuron where, x_t is the input and h_t the output. The loop (denoted by the recurring arrow) lets the information to pass from one step to the next step of the network.

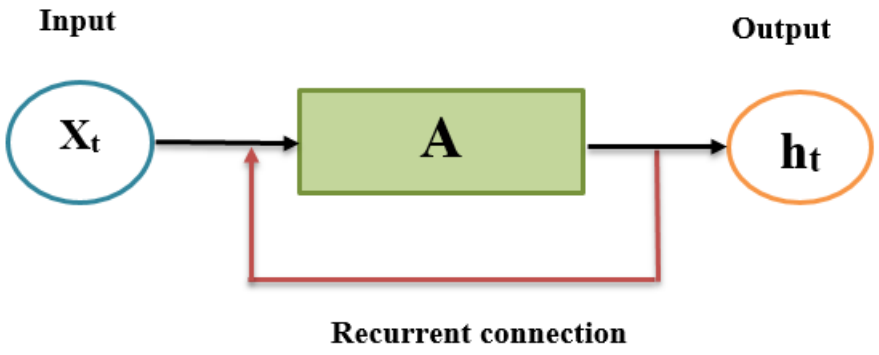


Figure 3.3: An RNN neuron

RNN hidden layers act as a memory unit. Precisely, the RNN output of time ***t−1*** effect the output of time ***t***. The RNN neurons are armed with feedback loops which yields the present output as the input for the subsequent step. The neurons of an RNN could be expressed like an internal memory which preserves the computational information from input in the previous step. For training an RNN, a variance of the back-propagation algorithm, termed as Back-Propagation-Through-Time (BPTT) is employed. Fundamental component of BPTT algorithm is a procedure called unrolling. Figure 3.4 illustrates the assembly of an RNN and the idea of unrolling. RNN can be unfolded in a graph without any cycles as presented in Figure 3.4, where, (X(t), X(t+1), …) represents multiple input time steps, (u(t), u(t+1), …) is multiple internal state time steps, and (y(t), y(t+1), …) as multiple time steps of outputs. When unrolling the RNN structure, and the internal state (u(t)) and the output (y(t)) of the prior time step are delivered as inputs to the subsequent time step.

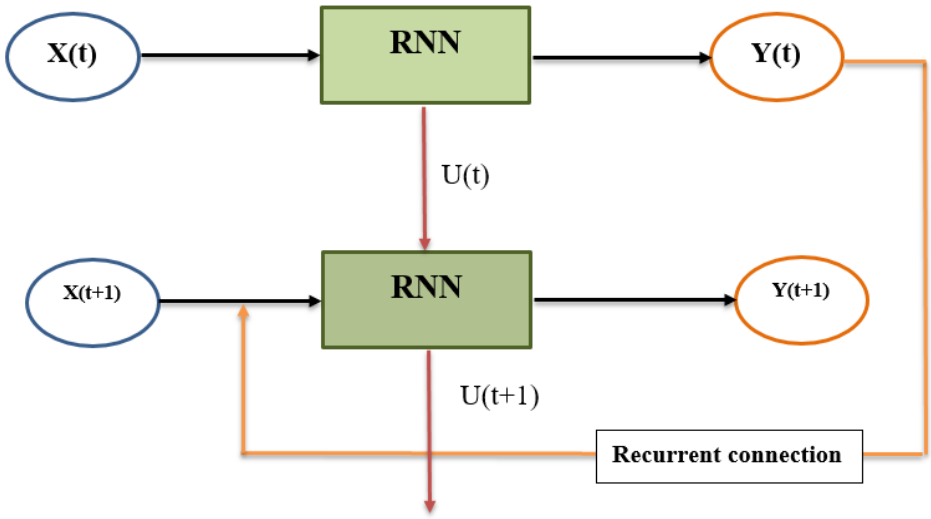


Figure 3.4: Unrolling of RNN architecture

RNN forms a class of powerful DNN that uses its internal memory along with loops for dealing with sequence data [48]. The unfolded architecture of RNNs in the Figure 3.5 represents the calculation procedure of an RNN unfolded (or unrolled) in time.

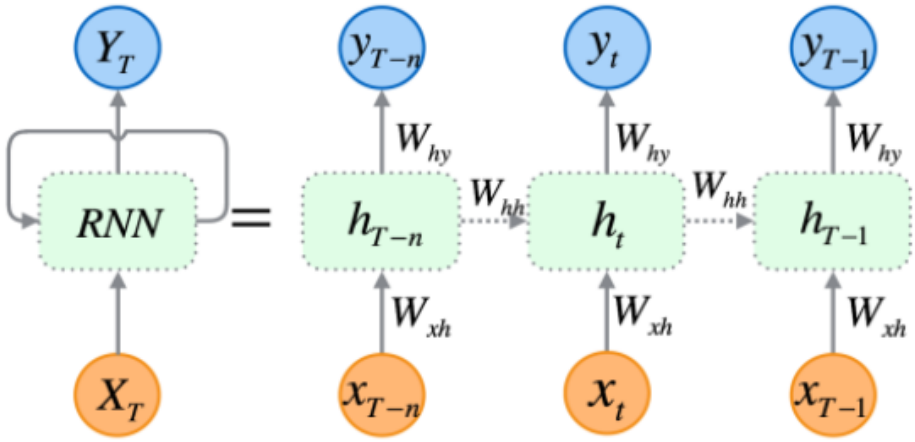


Figure 3.5: Unfolded RNN structure with T time steps [48]

In the above figure, during each iteration at time T, the hidden state of the hidden layer, h_T, gets updated depending on the current input X_T, and prior hidden state, h_{T-1}, through the following equation:

$$h_t = \sigma_h(W_{xh}X_t + W_{hh}h_{t-1} + b_h) \tag{3.3}$$

Where, W_{xh} represents the input layer to hidden layer weight matrix, W_{hh} denotes the weight matrix amongst two consecutive hidden states (h_{t-1} and h_t), b_h is the bias vector of the hidden layer, and σ_h denotes the activation function to generate the hidden state. The network output could be calculated as:

$$Y_t = \sigma_y(W_{hy}h_t + b_y) \tag{3.4}$$

Where W_{hy} denotes hidden layer to output layer weight matrix, b_y denotes the bias vector of the output layer, and σ_y represents the output layer activation function.

3.4. Long Short-Term Memory RNN

Long Short-Term Memory (LSTM) is an extension of RNNs. LSTM employs the idea of gates for its units. One major issue with RNNs is that it is unable to learn the context information across a prolonged span of time due to the vanishing gradient problem, which is, during a long temporal gap (i.e. time from when an input is obtained to the time when the input is used to make a prediction). Therefore, RNNs are incapable of learning from long-distance dependencies [28]. One answer to the problem of vanishing gradient is an LSTM design [28]. It averts the issue of the vanishing gradient and thus permits the retention of the elongated period of context information. Figure 3.6 shows an LSTM cell or neuron.

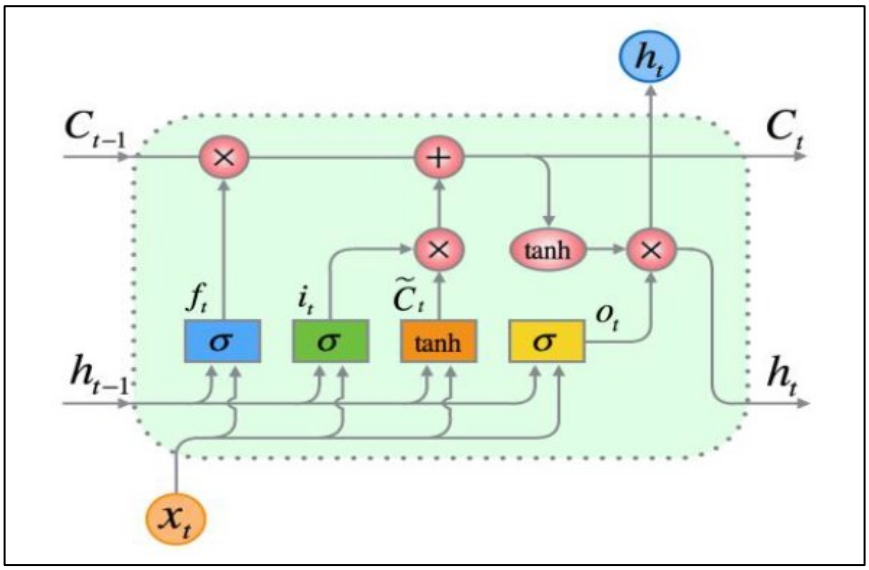


Figure 3.6: LSTM cell [48]

Since LSTM is an extension of RNN, the only additional component between RNN and LSTM architecture is the hidden layer [48] which is also referred as LSTM

cell. In Figure 3.6, at each time reiteration, t, the LSTM cell has input x_t, and the output h_t. During the training phase, the LSTM cell also considers the cell input state, $\tilde{C}_t$, the cell output state, C_t, and the previous cell output state, C_{t-1}. The gated structure allows LSTM to deal with aforementioned long-distance dependencies [48]. LSTM cell comprises of 3 gates, namely: input gate, output gate and forget gate. Figure 3.6 depicts the input gate, the output gate and the forget gate are denoted as i_t, o_t and f_t respectively at time t. All the three gates and the input cell state are denoted by colored boxes in Figure 3.6, are calculated by the following equations iterated from t = 1 to T:

$$f_t = \sigma_g(W_f X_t + U_f h_{t-1} + b_f) \tag{3.5}$$

$$i_t = \sigma_g(W_i X_t + U_i h_{t-1} + b_i) \tag{3.6}$$

$$O_t = \sigma_g(W_o X_t + U_o X_{t-1} + b_o) \tag{3.7}$$

$$\tilde{C}_t = \tanh(W_c X_t + U_c h_{t-1} + b_c) \tag{3.8}$$

Where W_f, W_i, W_o, and W_C denotes the weight matrices which maps the input of the hidden layer with the 3 gates and the input cell state, whereas the U_f, U_i, U_o, and U_C represents the weight matrices connecting the previous cell output state to the three gates and the input cell state. The b_f, b_i, b_o, and b_C are the bias vectors. The σ_g denotes the activation function of the gates, and the *tanh* denotes the hyperbolic tangent function. Based on the results of four above equations, at each time iteration t, the cell output state, C_t, and the layer output, h, can be calculated as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.9}$$

$$h_t = O_t * \tanh(C_t) \tag{3.10}$$

The last output of a LSTM layer would be a vector of all the outputs: $Y_T = [h_{T-n},\ldots,h_{T-1}]$.

3.5. Bi-directional Long Short-Term Memory RNN

The concept of Bi-directional Long Short-Term Memory (BLSTM) originates from bidirectional RNN [44] that processes sequence data in both frontward and backward directions using two distinct hidden layers. BLSTMs join these hidden layers with the same output layer. One inadequacy of traditional RNNs is that they are only capable of using the previous context. BRNNs [47] fix this by dispensing data in both directions. A BLSTM network computes the forward hidden layer sequence output , the output sequence of the backward hidden layer and the output layer ***y*** by reiterating the forward layer starting *t =1 to T,* backward hidden layer since *t = T to 1*, and then the final output is upgraded by the following equations:

$$\vec{h}_t = H(W_{x\vec{h}}X_t + W_{\vec{h}\vec{h}}\vec{h}_{t+1} + b_{\vec{h}}) \tag{3.11}$$

$$\overleftarrow{h}_t = H(W_{x\overleftarrow{h}}X_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \tag{3.12}$$

$$y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y \tag{3.13}$$

Both the output of the forward and backward layers are calculated by means of the standard LSTM equations, Equations (3.5) - (3.10). The BLSTM layer produces an output vector, $\boldsymbol{Y}_T$, which is calculated by the equation:

$$y_t = \sigma(\vec{h}_t, \overleftarrow{h}_t) \tag{3.14}$$

Where σ function combines both the output sequences. The σ function could be of four kinds: concatenating, summation, average and multiplication function. Incorporating BRNNs with LSTM neurons results a bidirectional LSTM recurrent neural network (BLSTM RNN) [45]. The BLSTM RNN is capable of accessing long-term context data in both the backward and forward directions. The combination of both the forward and backward LSTM layers is considered as a single BLSTM layer. It has been shown that the bidirectional models are considerably better than regular unidirectional models in various domains like phoneme classification and speech recognition [48]. Figure 3.7 illustrates a bidirectional LSTM structure with three consecutive time steps.

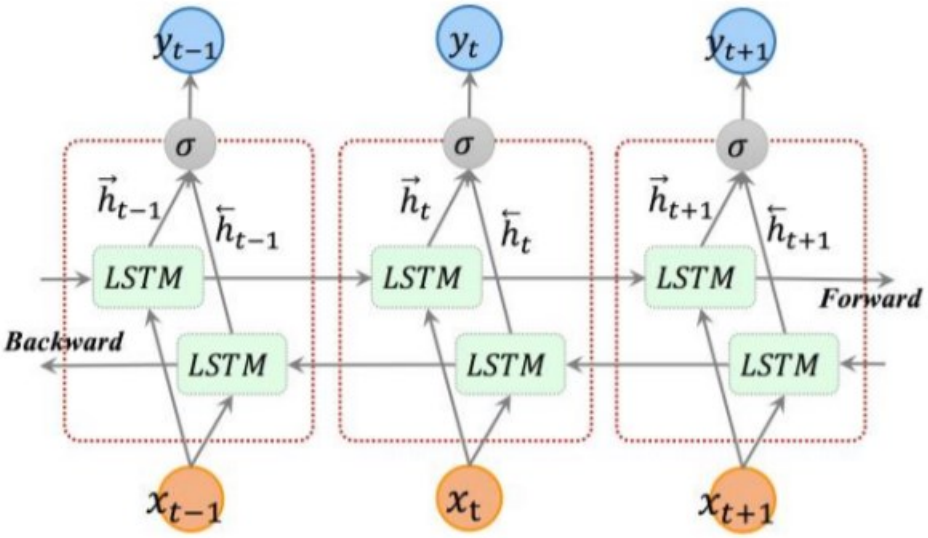


Figure 3.7: Unfolded BLSTM RNN structure with three consecutive time steps [48]

A thorough search of the relevant literature yielded no relevant paper which employs BLSTM for intrusion detection in IoT network. To fill this gap, a BLSTM RNN structure with the capability to deal through both frontward and backward dependences is introduced in this study. The BLSTM RNN architecture permits the integration of both previous and impending context through bidirectional optimization process.

3.6. Training Neural Networks

The implementation of neural network goes over two major stages: Training and Testing. During training, the NN is feed with knowledge (data) and the network is required to learn from its input data. The learning procedure is performed through an optimization (error minimization) process. Optimization algorithms are mathematical functions which help to reduce the loss function by fine-tuning the neural network parameters. The loss function computes the variance among the expected output and the actual output. Hence, minimizing the loss makes the network model generate optimal output. The optimization algorithm which is used for training the NNs is termed as Gradient Descent. The Gradient descent algorithm calculates the gradients or the slopes of the loss function with regard to the NN parameters (biases and weights). The technique which is used to calculate the gradients is termed as Back-Propagation (BP) [116]. The gradient is the amount of the alteration that occurs in the loss function due to the variation in the network parameters. Depending on the gradient the network parameters are updated by means of a scalar value called learning rate.

This mechanism is performed via iterations by allowing several repetitions over the training data. One surpass over the training data is known as an epoch. Since each epoch, the parameter values move nearer to the optimal value resulting in the loss function convergence. For a large dataset computing the loss and gradient for the full dataset might be computationally infeasible. Hence, a variance of the gradient descent known as Stochastic Gradient Descent (SGD) is widely in use. In SGD algorithm, the total input is distributed into smaller subsets of input termed as batches. NN parameters are then updated by computing the loss function of single batch at a time. There are several other popular variants, namely: RMSprop, AdaGrad and Adam [117].

Training the NN is often associated with the problem of overfitting, which is, when the network is characterized with high accuracy over the training-set but generates poor accuracy when evaluated on a new test data. Several counter measures can be applied for preventing overfitting. One is Early-Stopping, where the loss function of a validation set (a small sub-set of training set) is calculated after each epoch. If the value of the loss function over validation set starts increasing, despite the decreasing loss of the training set, it could be a sign of overfitting. In that case the training should be stopped. Another technique is dropout regularization, which is frequently used in deep learning where a certain ratio of neural network connections are eliminated randomly over each epoch. The network weights and biases gets updated by the training algorithm like Back-Propagation (BP), Back-Propagation-Through-Time (BPTT) etc. Parameters like dropout, decay, batch size, learning rate etc. are the optimization algorithm parameters which are generally determined by the researcher over trial-and-error. All these parameters are collectively called as hyper-parameters.

3.7. Activation Function

Neurons are the building blocks an ANN. Neurons take inputs from the preceding neurons, multiply the input values with weights, generate a sum of products, and pass the summation through an activation function to generate the final output (3.16). Mathematical illustration of the neuron is presented in (3.15).

$$Y = \sum (input) * (weight) + bias \tag{3.15}$$

$$Output = f(Y) \tag{3.16}$$

Firing a neuron actually means activating it. The similarity between biological neurons and artificial neurons is illustrated in Figure 3.8.

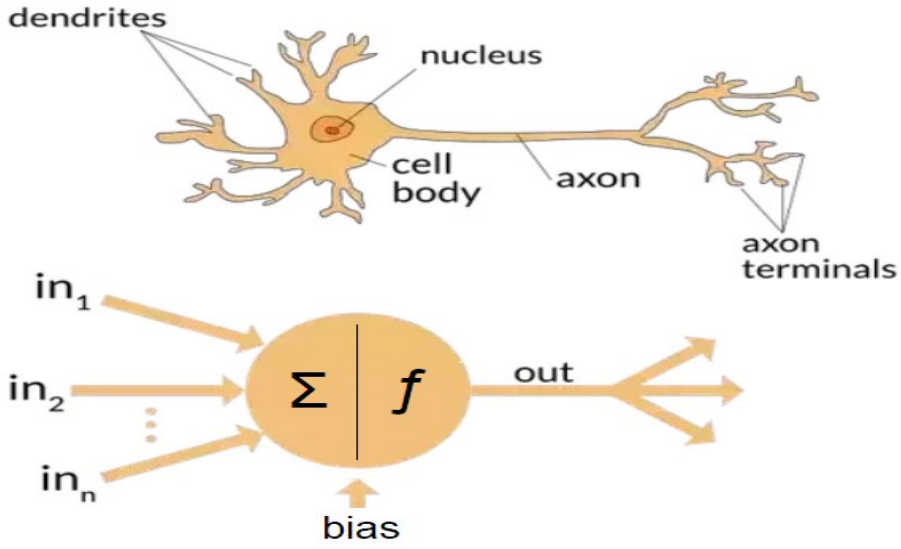


Figure 3.8: Biological neuron and artificial neuron [122]

In the above Figure 3.8., the dendrites carry the electrical signals to the neuron body and act as the neuron inputs. Similarly, in an artificial neuron, the inputs in_1, in_2,…, in_n resembles the dendrites. The activation function resembles the cell body, and the propagated output resembles biological axon. The artificial neurons imitates a similar functioning logic as that of a biological neuron.

3.7.1. Step Function

It is a NN activation function which receives *f(x)* as input. Figure 3.9 illustrates the Step function diagram, where X represents the threshold value. If *f(x)* exceeds a definite given value (called threshold), the step function output is fired (i.e. activated), else the output of the Step function remains non-activated [123].

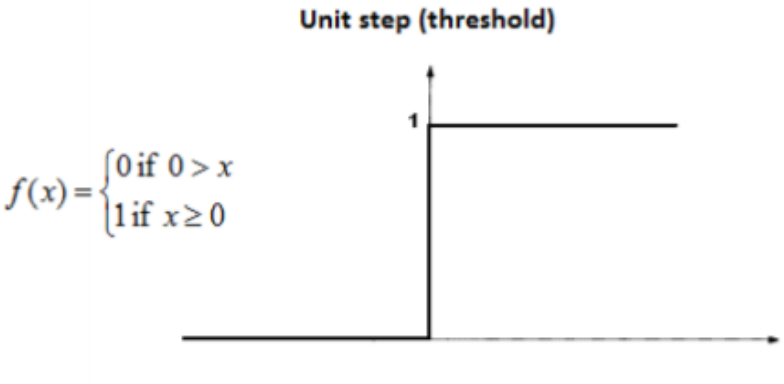


Figure 3.9: Step function [123]

There are several step functions that are widely used in machine learning. For instance: Sigmoid function, Tanh (hyperbolic tangent) function and Rectified Linear Unit (ReLU) function. Among these, ReLU is the most popular step function in the area of RNN. A brief discussion of ReLU step function is provided in the following section.

3.7.1.1. Rectified Linear Unit (ReLU)

This function is the most broadly employed solution for vanishing gradient issue of LSTM RNN. The mathematical representation is shown in (3.17).

$$Y = f(x) = \max(0, x) \tag{3.17}$$

When the input remains smaller than 0, then output remains 0. When the input is greater than 0, then the input and output becomes equivalent. The ReLU function is more efficient for a binary classification problem, and we employ it as hidden layer activation functions in our proposed model.

3.8. Deep Learning

Neural network layers are comprised of neurons. The number of units or neurons that constitute the input layer (i.e. the first layer of the network) would be equivalent to the amount of data-set features [53], it's not mandatory though. The output layer should consist of only one neuron along with an activation function while solving a binary classification problem [53]. The third type of layer that neural networks have are called hidden layers, which is placed in the middle of the input and output layer. The quantity of hidden layers determines the depth of the NN. A NN with

lots of hidden layers are termed as Deep Neural Network (DNN) and Deep Learning (DL) represents the learning algorithm of the DNN. Salient features of DL is that it can learn the features by itself, thus there is no need of hand crafting the features. This DL property facilitates the learning procedure and makes DNN more efficient and robust in comparison to the shallow learning [53].

3.9. Dropout Regularization

Deep neural network (DNN) models have numerous parameters and have the ability to model highly composite functions. This capacity is a boon and a bane. Such prototypes would frequently overfit on the training-set and would drop accuracy and generalizability over the test-set [56]. Regularization in ANN terminology speaks of the procedure of regulating neural network layers for preventing the over-fitting. Dropout (also known as dropout probability or dropout rate) is the most widely utilized regularization technique in DL. During the learning process the hidden layer(s) neurons are selected randomly and are discarded depending on the dropout rate. Precisely, randomly selected neurons are dropped-out i.e. dropped out neurons could not update weights any more, thus helping the learning process to evade the problem of overfitting [55].

3.10. Deep Learning Loss Function

In a NN, the corresponding neuron weights are tuned via back-propagation algorithm. Mathematical representation of BP is shown in (3.18).

$$\Delta W_t = \alpha * \frac{\partial MSE}{\partial W_t} + \mu * W_{t-1} \tag{3.18}$$

Where, W_t represents the weight change of a specific edge at time ***t***. W_{t-1} represents the weight change of the prior iteration. The learning rate is denoted by α. And Δw_t signifies the total weight difference at time ***t.*** The Δw_t is computed by employing a loss function. There are several loss functions that are used to calculate the loss. In the following sections some of the popular loss functions are discussed.

3.10.1. Mean Squared Error

MSE or Mean Squared Error loss function remains one of the most widely used in the area of DL. The mathematical function of MSE is presented in (3.19).

$$MSE = \frac{1}{n}\sum_{n}^{i=1}(\hat{y}_i - y_i)^2 \tag{3.19}$$

Where, Y_i forms the learning procedure output, $\hat{Y}_i$ is the expected output and n is number of output classes.

3.10.2. Cross Entropy Loss

Also termed as CEL, is one more widely used loss function which is frequently chosen for regression or classification issues. Mathematical representation of CEL is presented in (3.20).

$$cross\_entrophy = \sum_{k}^{i=1} y_i \log(\hat{y}_i) \tag{3.20}$$

Where, i denotes the amount of training instances, $\hat{Y}_i$ is the expected outcome and Y_i is the learning output [56]. CEL and MSE are extensively employed in classification problems.

3.11. Deep Neural Network Implementation Frameworks

Use of deep learning architectures has grown fast, and this growth has been sustained by various deep learning frameworks in current years. Every DL framework has their own strength and weakness depending on the optimization algorithms, DL architectures, and convenience of deployment and development [92]. Many of these DL frameworks has been widely used in research work for proficient implementation of DNNs. In the subsequent sections some of the frameworks are reviewed.

3.11.1. H2O

H2O framework supports interfaces for Java, JSON, Python, CoffeeScript/JavaScript, R and Python [93]. It is capable of executing in several different modes like on Hadoop, on Spark Cluster and standalone mode. H2O is capable of accommodating both ML algorithms and DL algorithm.

3.11.2. Torch

Torch provides an easy to access platform for ML comprising a variety of deep learning algorithms which facilitates easy deployment of deep neural network models [95]. It is an open source light weight DL framework based on Lua programming language and supports faster training of the ML models. It also supports ML model development for both GPUs and CPUs, and provides efficient parallelization packages in order to train the DNNs.

3.11.3. TensorFlow

TensorFlow was originally intended for Google Brain project. It is employed by various Google merchandises like Google Translate, Google Search, Google Maps and YouTube. It is an open source library for ML implementation which uses several types of DNNs [94]. Tensorflow implements graph representations for building NN models. TensorFlow also comes with a visualization package called TensorBoard, where, NN models and their learning procedure could be observed.

3.11.4. Caffe

Caffe is a C++ based DL algorithm which supports CUDA for GPU computation. Caffe is open source and also supports Matlab and Python interfaces. Model representation and implementation are separated in Caffe. It defines models by configuration rather than hard coding in the source code, which in turn makes the switching between the platforms possible (for example, GPU to CPU or other mobile devices) [97].

3.11.5. Theano

It is a competent ML framework based on Python programming language for that supports GPU and CPU compiling. Theano employs graph representations and utilizes the CUDA library for code optimization on GPUs. It is an open source ML framework that permits parallelism on CPUs. Several wrapper libraries like keras, Pylearn2 and Lasagne convey simpler programming interface over Theano [96].

3.11.6. Neon

Neon is a Python-based open source DL framework that exhibits high efficiency for contemporary DNNs like AlexNet, GoogleNet and VGG. It supports the development of various widely used NNs, like AEs, RNNs, CNNs and LSTMs [98]. Neon is capable of operating on both GPUs and CPUs and also offers easy shifting of the backend platform of hardware. A study done by Bahrampour *et al.* [99] shows comparative investigation of the previously mentioned frameworks. In spite of the fact that the productivity of each framework varies depending upon situations, Theano and Torch showed better overall performance. A summary of comparison of several DL implementation frameworks are presented in Table 3.1.

Table 3.1: Assessment of different DL implementation framework [39].

Framework	**Core Language**	**Interface**	**Pros**	**Cons**
TensorFlow	C++	C, Java, C++, Python and Go	• Fast on LSTM training procedure • Supports visualizing networks	• Slower training process related to other Python-based frameworks
H2O	Java	R, Python, Scala, REST API	• Extensive range of interfaces	• Number of supported models are restricted. • Nonflexible
Torch	Lua	C++ and C	• Supports several models • Upright documentation	• Learning a new programing language

			• Comprehensive error debugging	
Theano	Python	Python	• Several models are supported • Rapid training of LSTMs over GPUs	• Various low level APIs
Neon	Python	Python	• Quick training time • Platform switching is easy • Provisions modern architectures like GAN	• Not supportive for CPU multi-threading
Deeplearning4j	Java	Python, Scala, Clojure	• Models get imported from leading frameworks (such as Theano, Caffe, Torch and TensorFlow) • Provides visualization interface.	• Extended training time related to other tools
Caffe	C++	Python, Matlab	• Offers a bunch of models as reference • Elementary platform swapping • Excellent for CNN	• Not very good for RNN

3.12. Training Dataset and Feature Identification: *UNSW-NB15*

Machine learning (ML) and the techniques of data mining are extensively utilized for advance detection of intrusion in contemporary years which makes it probable to automate intrusion detections in IoT network. One of the major research difficulties that IoT intrusion detection research face is the inaccessibility of a comprehensive network-based dataset which mirrors modern network traffic environment [23]. Several current researches exhibited that for the present network threat scenario, those datasets do not conclusively reflect modern network traffic and contemporary low footprint attacks. In [20], Bajaj and Arora express that the KDD dataset is obsolete, recommending that the NSL-KDD dataset is most appropriate for analyzing current networks. They also stress that, KDD 99 dataset endures with redundant information that leads to biased outcome of intrusion detection. This results in inappropriate feature classification. They also claim that intrusion detection studies would likely generate results that do not characterize the real network scenarios due to the use of KDD 99 datasets [20]. However, the absence of alternatives is the reason the dataset is still being used. However, answering the unobtainability of network benchmark dataset encounters, in the year 2015, Moustafa and Slay [23] came up with their studies and produced UNSW-NB15 dataset. The authors claim that the introduced dataset comprises a fusion of the contemporary real network traffic and synthesized threat actions. The authors castigated that NSL-KDD dataset and KDD'99 dataset don't characterize the up-to-date interventions in IDS and presented a comprehensive and all-inclusive dataset called the UNSWNB15. This dataset encompassed several features from KDD'99 dataset [24]. They further inspected the features of UNSW-NB15 dataset and KDD'99 dataset and the results attained exhibited that the actual KDD'99 dataset attributes are less effective compared to UNSW-NB15 features. The new UNSW-NB15 intrusion dataset that includes diverse attributes or features including those in the KDD'99 dataset. This newly generated UNSW-NB15 intrusion dataset forms the most up-to-date dataset, published in 2015 to facilitate intrusion detection research works.

Another challenge that this area of research face is obtaining the labeled input dataset for the purpose of intrusion detection in IoT [5]. Fu *et al.* [5] recognizes out this difficulty that majority of the independent researchers face in the field of IoT security. For overcoming the challenge, UNSW-NB15 dataset have been employed in this research work. Moustafa and Slay (in 2015) [23] suggested that the NSL-KDD dataset and KDD'99 dataset did not characterize the up-to-date features for intrusion

detection, and presented a comprehensive and all-inclusive dataset called the UNSWNB15. This dataset encompassed several features from KDD‘99 dataset. In [9], they further analyzed the features of the UNSW-NB15 and KDD‘99 dataset. Results demonstrated that actual KDD‘99 dataset features were less representative as compared to the features of UNSW-NB15 intrusion dataset. The UNSW-NB15 dataset contains 45 features [23]. The dataset is further split into separate training and testing set containing all the current attack types. T. Janarthanan and S. Zargari [26] performed an extensive study on the UNSW-NB15 dataset for extraction of most competent features and thus proposed a feature subset which dramatically increased the intrusion detection efficiency. In this thesis, the dataset subset in the file *UNSW-NB15_training-set.csv* will used for training the proposed model, while the *UNSW_NB15_test-set.csv* will be utilized for testing the proposed model. Both the dataset files can be obtained from: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

The data set file *UNSW-NB15_training-set.csv* contains 175,341 records for training, while the test set file *UNSW-NB15_testing-set.csv* contains 82,332 records. the UNSW-NB15 dataset has 9 attack types in total, out of which 5 types are often present in IoT attacks (Analysis, Backdoor, Denial-of-Service, Worms, and Reconnaissance) [118][119]. Hence, we extracted only these 5 types of attack samples along with the 'normal' samples to prepare our training-set. In [4], the authors have used UNSW-NB15 dataset for conducting IoT research because unlike previous benchmark datasets, UNSW-NB15 exhibits contemporary attack patterns and modern normal traffic patterns. Moreover, since UNSW-NB15 has separate training-set and testing-set, data distribution remains different [4]. Again, in [31], the authors points out that: “It encompasses realistic normal traffic behavior and combines it with the synthesized up to date attack instances”. Also, [35] points out that previous benchmark data sets like KDD‘99 and NSL-KDD could not meet the current network security research needs as they does not comprehend the present-day network security circumstances and the latest attack features.

The UNSW-NB15 data set have been chosen for this research purpose as it covers modern attack patterns, consists of modern normal traffic patterns, and contains only two classes (“attack” and “normal”). Since we are performing binary classification task, this class distribution facilitates the approach that has been proposed. Secondly, UNSW-NB15 forms a comprehensive data set that presents 5 types of IoT attacks. Thirdly, UNSW-NB15 is a sequential dataset, which is very appropriate for training recurrent neural networks.

3.13. Dataset Format Conversion

The training set is named as *NSW-NB15_training-set.csv*. The training set has 9 types of attack, out of which 5 types of attack are often present in IoT attacks [118][119]. In order to make the dataset suitable for this research work, samples of only those 5 attack types are extracted. The UNSW-NB15 training-set has 45 features in total. For this research purpose, the attribute sub-set proposed by [9], namely, *service, sbytes, sttl, smean, ct_dst_sport_ltm* are considered as it reduces the problem of overfitting, reduces training time and improves the accuracy [9]. Hence, the training-set which is used for this research work consists of 5 attack types and 5 features. A total of 5450 samples of data are randomly selected from *UNSW-NB15_training-set.csv* and placed it in a new .csv Microsoft Excel file named *UNSW_NB15_training-set_5450.csv*. The approach of Fu *et al.* [5] is followed by manually manipulating the training dataset, where, the authors has followed the approach of manually adding some abnormal samples within the dataset for making the dataset fit for their research purpose. It is the advantage of the approach that the input dataset would be appropriate for intrusion detection, which would fit the goal of the research. Moreover, the approach helps in managing the issue of procuring labeled intrusion datasets at a high cost. Here, the approach of manual manipulation is followed and the above mentioned 5 features (*service, sbytes, sttl, smean, ct_dst_sport_ltm)* along with 5 attack types (*Analysis, Backdoor, DoS, Reconnaissance and Worms*) are extracted manually. Thus, the resulting training-set (*UNSW_NB15_training-set_5450.csv)* consists of 5 types of attacks, 5 features and two class labels: “Attack” and “Normal”. Table 3.2 shows the resulting data-set structure of the training-set.

Table 3.2: Data-set structure after extracting the features manually

service	sbytes	sttl	smean	ct_dst_sport_ltm	attack_cat	label
-	4238	31	59	1	Normal	0
-	3752	31	208	1	Normal	0
-	1280	254	52	1	DoS	1
smtp	37310	31	718	1	Normal	0
-	168	254	84	1	Reconnaissance	1

Table 3.2 shows five rows of the dataset after format conversion. First five columns represent the five selected features proposed by [9] and their corresponding

values. Column six (attack_cat) shows the category of attack including normal samples, and the last column shows the dataset label, where, value 0 is equivalent to normal samples and value 1 represents attack samples.

Chapter 4

RESEARCH METHODOLOGY

4.1. Introduction

This chapter discusses the research methodology of this project. It comprises of the description of the data preprocessing methodology and implementation methodology. This chapter also describes the methods we have followed to develop our proposed BLSTM RNN classification model. This chapter also defines the evaluation criteria, i.e. the conditions by which the model's detection accuracy in IoT could be measured.

4.2. Model Design Methodology

The proposed model aims to detect intrusions at the transport layer of the IoT architecture. In order to detect intrusions in the data at the IoT transport layer, a process design is required which would accept IoT data as input, perform the processing and generate a two-fold classification: "attack" or "normal". A high level model design view is shown in Figure 4.1.

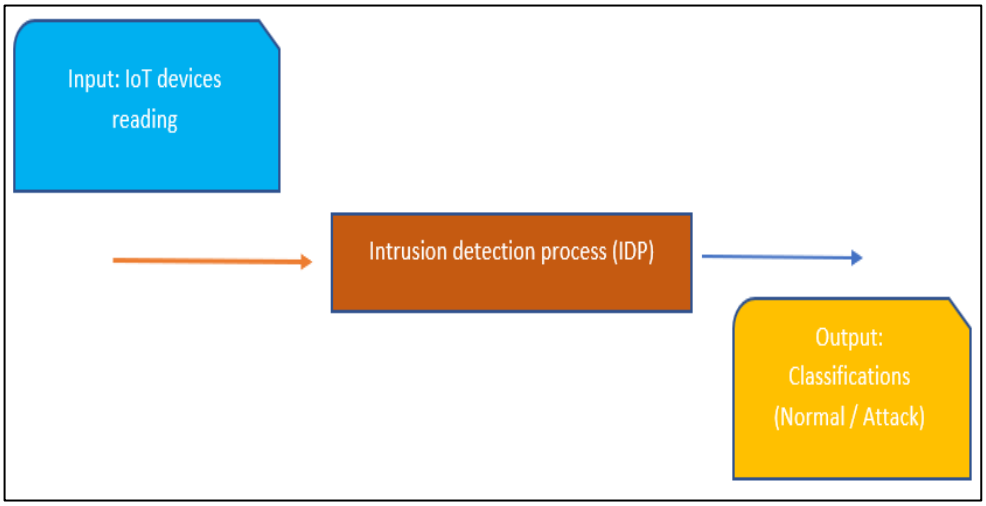


Figure 4.1: Intrusion Detection Process (IDP) – high level view

The Intrusion Detection Process (IDP) is responsible for the intrusion classification task. It consists of three stages. Figure 4.2 illustrates IDP in further depths.

1. **Preprocessing Input Data:** This phase deals with the conversion of input data in an acceptable data structure permitted by the simulation framework.
2. **Training:** This phase includes fitting the NN model with the training data for classification.
3. **Detection**: In this stage the trained NN model performs the detection of intrusions.

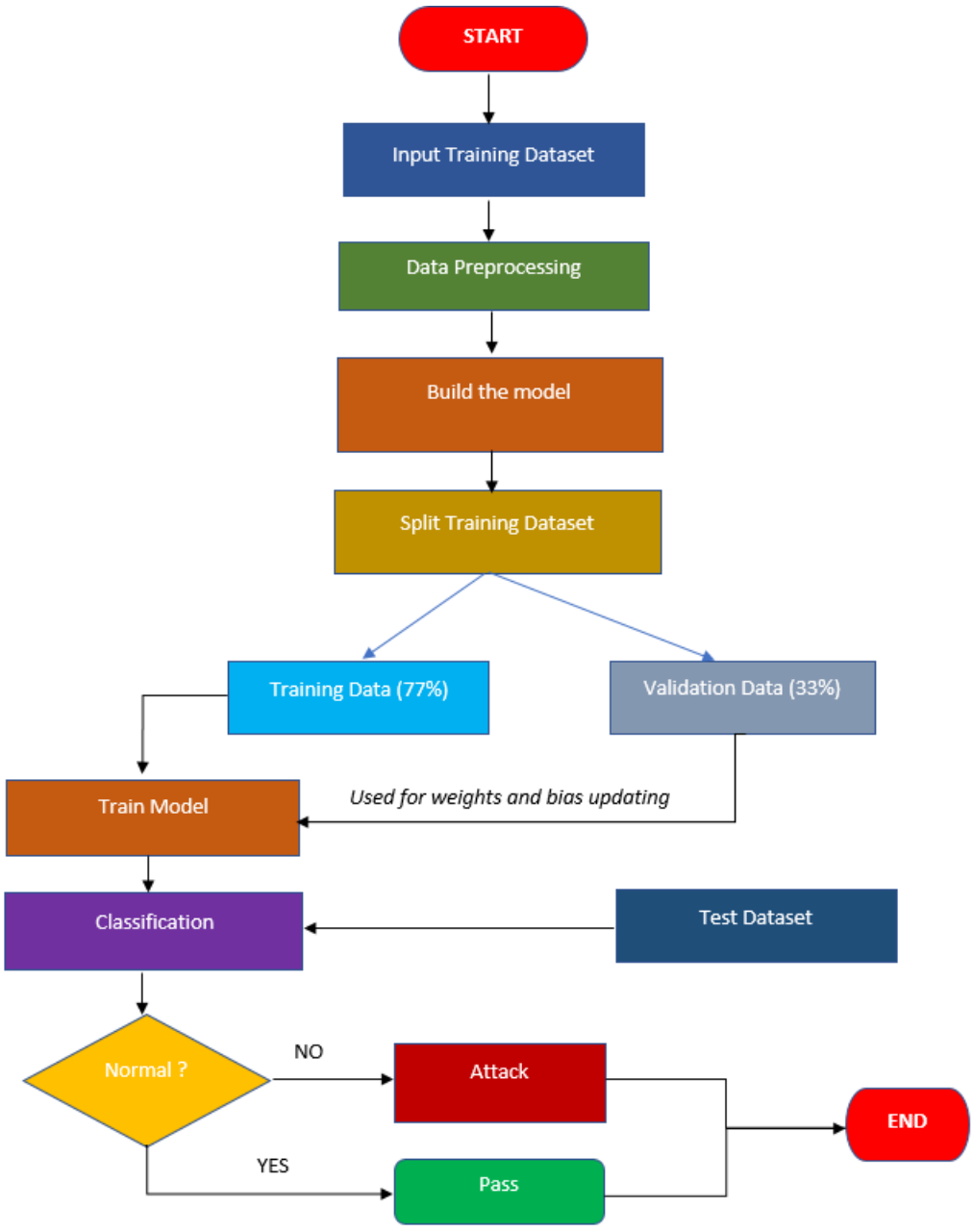


Figure 4.2: IDP Flowchart

In Figure 4.2 above, during the preprocessing stage the training-set data samples are encoded and normalized and fit into a data structure compatible to the

TensorFlow framework. After preprocessing, the whole dataset will be divided into two subsets: Training subset and validation subset. The former one, which is the training subset is used for training the BLSTM RNN network, and the validation subset is used for validating the already trained network. After the completion of the validation process a separate testing dataset is used for testing the model classification accuracy and other performance measures.

4.3. Implementation Methodology

4.3.1. Keras Library

In this thesis, Python programming language and Keras library will be used for the implementation purpose. Keras is a neural network library which is open source and offers a high-level Application Programmer Interface (API) for implementing DNNs. Keras executes atop several other DL frameworks such as TensorFlow[1] and Theano[2]. In this research work, we will be using Keras atop TensorFlow. Keras is chosen as Keras API enables rapid prototyping of neural network models in research. Keras also permits modular configuration of NNs, i.e. it allows to combine parameters like activation functions, loss functions and optimizers. Moreover, the Keras API is easy to learn and use, and has the added advantage of easily porting models between frameworks. Since Keras is self-contained, it can be used without having to interact with the back-end NN framework, which is TensorFlow in our case. This approach minimizes the complication and need for programming the back-end framework and enables fast experimentation. We have chosen Keras, primarily, for the following advantages:

- Better user experience for deep learning algorithms: The Keras API is user friendly. The API is well designed, object oriented, and flexible. Researchers can define new deep learning models without needing to work with potentially complex back ends, resulting in simpler and leaner codes [40].
- Persistent Python integration: Keras is a native Python package, which allows easy access to the entire Python data science ecosystem. For example, the Python Scikit-learn API can also use Keras models [40].

[1] https://sheffieldml.github.io/GPyOpt/

[2] http://deeplearning.net/software/theano/

- Portability: Keras allows researchers to port from Tensorflow back-ends to other back-ends like Theano. In addition, Keras makes many learning resources, documentation, and code samples freely available [40].
- We have used Keras library as it takes into account simple and quick prototyping through modularity, extensibility and user friendliness extensibility [40].

4.4. Evaluation Methodology

To characterize the efficiency and detection accuracy of the proposed model, confusion matrix would be utilized. The confusion matrix is a 2-dimensional matrix representing the correlation amongst the detected and actual values as shown in Figure 4.3. True Positive (TP) specifies the count of anomalous or unusual samples that are accurately identified by the model. False Positive (FP) signifies the count of samples that are labeled as normal in the dataset but are recognized as anomalies by the model. True negative (TN) signifies the amount of normal samples that are detected as normal by the system. False Negative (FN) refers to the amount of attack samples which have been labeled as normal by the model.

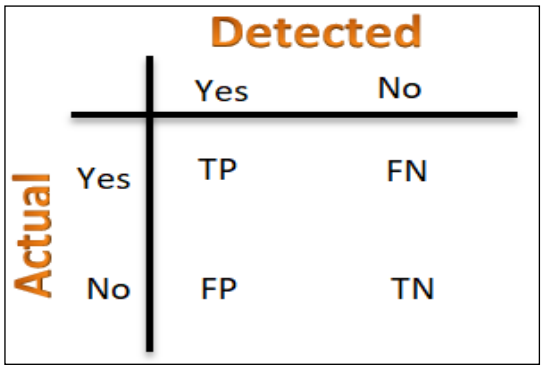


Figure 4.3: Confusion Matrix.

The overall accuracy of the model, could be precisely defined as – how frequently the model is correct. The overall accuracy of the model is computed by (4.1):

$$Accuracy = \frac{TP + TN}{X} \quad 4.1$$

Where, X denotes total number of samples present in the input dataset.

The model's misclassification rate could be defined as – how frequent the model is wrong. Misclassification rate is the percentage of wrong detections and can be calculated by using the formulae in (4.2):

$$miscalculation\_rate = \frac{FP + FN}{X} \tag{4.2}$$

False Positive Rate (FPR), calculated by (4.3), is the percentage at which the system incorrectly classifies normal samples as anomaly:

$$FPR = \frac{FP}{X_{Normal}} \tag{4.3}$$

Where, X_{Normal} is the number of actual normal samples in X.

Other parameters for evaluating the proposed model includes recall, precision, and f1-score values Precision is calculated as the ratio of correct positive detections to the total actual positive detections, as shown in (4.4):

$$precision = \frac{TP}{TP + FP} \tag{4.4}$$

Recall is the ratio of correct positive detections to the number of actual abnormal samples, as presented in (4.5):

$$recall = \frac{TP}{TP + FN} \tag{4.5}$$

In (4.6), F1-Score denotes the harmonic mean of recall and precision. F1-Score is calculated by the weighted average of precision and recall by taking both the FP and FN into account.

$$f_1 - score = \frac{2(recall * precision)}{recall + precision} \tag{4.6}$$

These metrics are employed to evaluate the proposed model in the testing phase of the model simulations.

4.5. Hardware and Software Used

Google's TensorFlow framework will be used to perform the neural network experiments. A powerful python library called Keras will be used to build and implement our proposed BLSTM RNN model. The experiments are conducted in the below mentioned environment:

CPU: Intel ® Core ™ i7-7500U CPU @ 2.70 GHz
RAM: 16GB
OS: Windows 10
Programming Language: Python
Libraries used: numpy, scikit-learn, keras, pandas, and Tensorflow.

Chapter 5

ARCHITECTURE & IMPLEMENTATION

The model architecture is discussed in this chapter, along with the implementation of the proposed BLSTM RNN intrusion detection model. The proposed model would be implemented through Python programming language and Tensorflow as the back-end implementation framework. *Keras*, the neural network library written in Python is used, which operates atop the Tensorflow framework. The use of Keras makes the implementation process scalable, fast and straightforward. Several other libraries are also utilized, namely: *Pandas* and *NumPy*. The model will be implemented in Spyder (a scientific interactive development environment for Python language) using the Tensorflow library. The whole implementation process is divided into three major phases: data pre-processing phase, model training phase and lastly, model testing phase. This chapter is organized as: Section 5.1 explains the architecture of the proposed model. In Section 5.2 programming language and libraries used in implementing the model is discussed. It is followed by the details of implementation in section 5.3.

5.1. System Architecture

As depicted in the Figure 5.1 below, the proposed classifier is a layered BLSTM RNN architecture with one input layer, three hidden layers which are densely connected, and one output layer. The input layer comprises of five BLSTM units. The first, second and third hidden layer of the proposed architecture has 220,240, and 260 BLSTM units respectively, and the dense output layer consists of 1 neuron. The model will employ Adam as network optimizer with a decay of 0.99, a learning rate of .05, and a batch size of 132. The model will be iterated for 100 epochs. To avoid the model to overfit the training data because of excessive training epochs, the dropout regularization technique have been used (mentioned in Section 3.9). Other hyper-parameters of the proposed architecture is shown in Table 5.1.

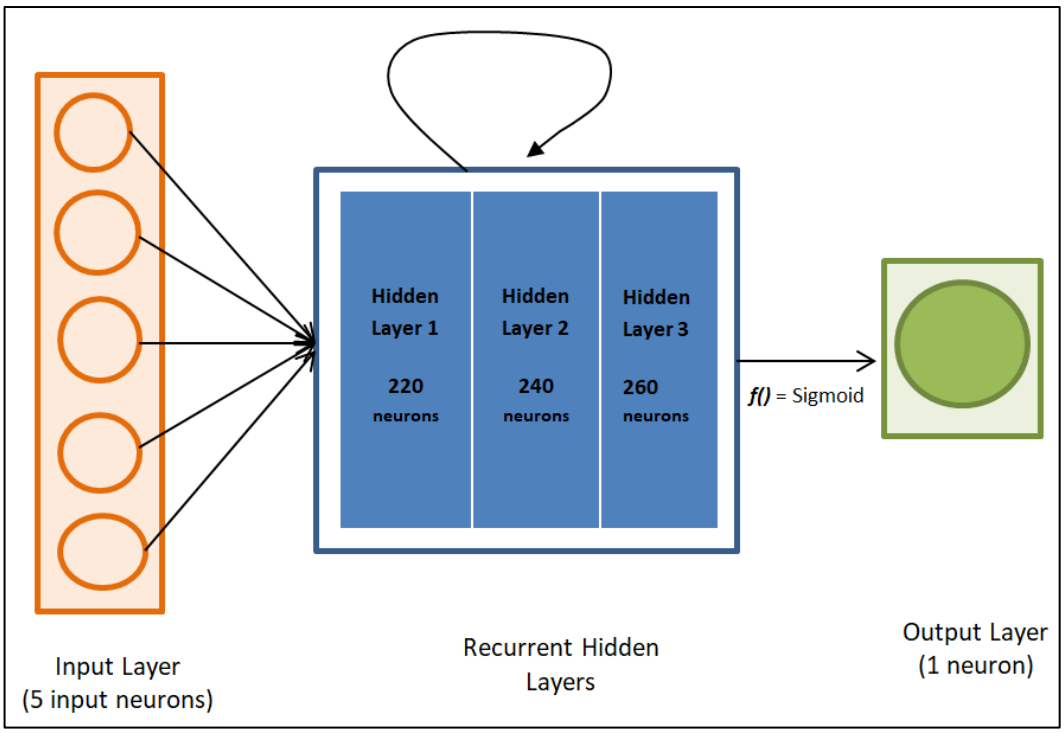


Figure 5.1: System Architecture

Table 5.1: Design parameters of the proposed model

Epoch	**100**
Hidden Layers	3
Activation function	Sigmoid
Optimizer	Adam
Classification engine	binary_crossentropy
Learning Rate	0.001
Class size	2
One hot encoder	Yes
Weights	Random
Biases	Random

5.2. Technical Knowhow – Programming Language, Development Environment, Backend Framework and Libraries

The Python programming language have been used for implementation purpose, as Python provides superior quality data science libraries with expedient development environment: Spyder - which is a scientific interactive development environment and is great for easy and fast visualization. It has many integrated features that facilitates RNN implementation, and it is super easy to install. Spyder provides

many advanced Graphical User Interface (GUI) functionalities that assist for RNN implementation. For instance, the "Variable Explorer" GUI helps to visualize the variables (data and values) used in the implementation. We have used this functionality to visualize and analyze the dataset values, confusion matrix threshold values, prediction metrics, etc. Moreover, it facilitates the running and debugging of the python code through syntax coloring and breakpoints. Spyder IDE also supports parallel-run, i.e., multiple neural networks can be trained and/or tested simultaneously. While conducting our experiment, this feature have been used to train and test several RNNs simultaneously. Spyder IDE integrates the essentials libraries for developing RNN, like, NumPy, SciPy and Matplotlib.

For implementing the BLSTM RNN, the high level neural network library called Keras [116] have been employed. The Keras *Sequential* class is utilized for instantiating the RNN object. Other Keras classes like *LSTM* and *Bidirectional* ia being used for implementing the proposed BLSTM model. *Dropout* is another class that belongs to Keras library which have been used for preventing network overfitting. In our implementation, we use the Google TensorFlow as backend neural network framework. For manipulating the matrices effectively, the *NumPy* and *Pandas* [117] libraries are employed. *NumPy* is primarily used to create the Tensorflow data structure. *NumPy* also allows to use much less memory for matrices than the default python lists, and it also makes the matrix operations much more efficient. *Pandas* is built on top of *NumPy* and it provides higher level interface for manipulating datasets with named rows and columns (the input datasets are required to be *Pandas* dataframes). We have used *Pandas* library to import the values from training-set and test-set (which are .csv files) and store them as data frames. For measuring the performance of the detection system, some helper functions have been used from the library sklearn [115] namely: confusion_matrix and classification_report.

5.3. Code Structure

Since python is an object oriented programming language (OOP), objects (classes) have been employed to implement the functionality of the layers described in section 5.1. Figure 5.2 below depicts the interaction among the classes and their respective methods. The *build()* method of the IDS class instantiates the other four classes (Data class, Classifier class, FitModel class and Detection class), so that they can be later used in the method *execute()*. The methods are called in the order as the

arrows suggest. The '*param'* signifies the function parameter. *X_train* and *Y_train* are the training-set matrix. *X_test* is the test-set matrix.

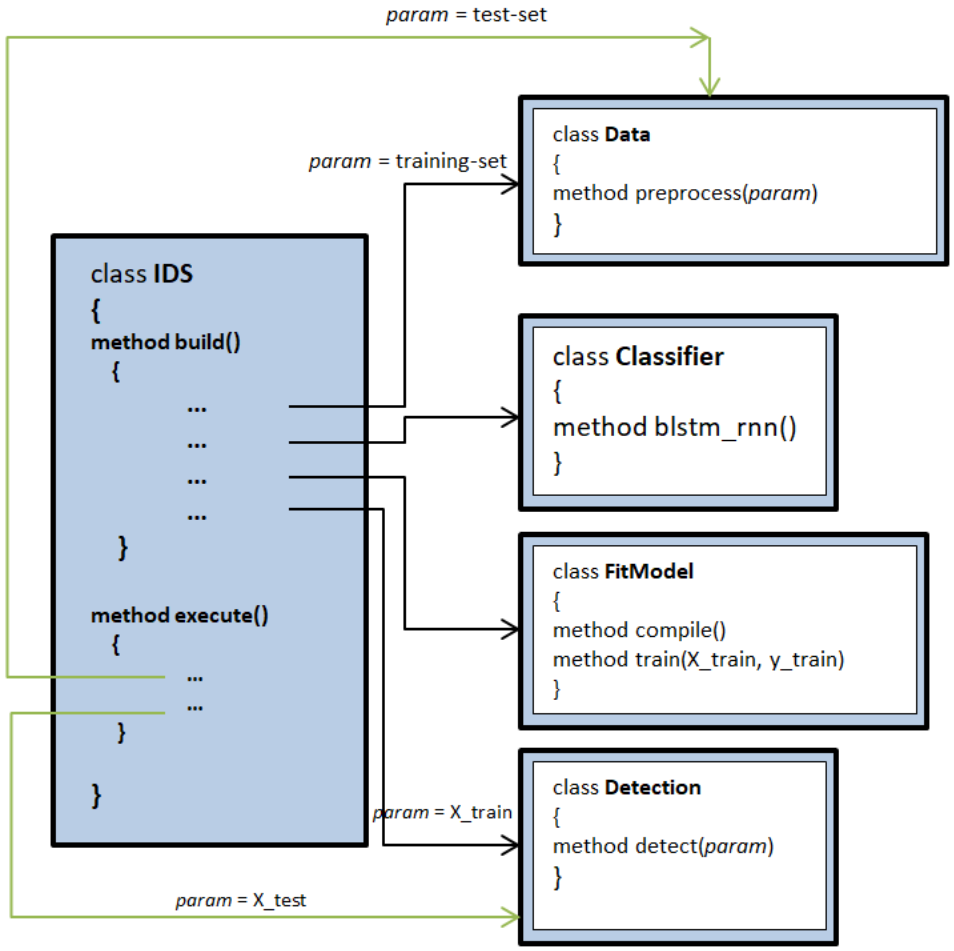


Figure 5.2: Scheme of Implementation.

As in Figure 5.2, five different classes have been developed, i.e., IDS, Data, Classifier, FitModel, and Detection, in order to implement our system architecture. Table 5.2 summarizes functions of all the used classes. In the following sections, the classes and their respective methods are described in detail.

Table 5.2: Classes and their respective functions

Class Name	Description
IDS	The IDS class is the parent class which instantiates and encapsulates other four classes and their respective methods.
Data	The Data class deals with the data preprocessing mechanism.
Classifier	This class builds the code for the BLSTM RNN architecture.
FitModel	This class is responsible for two tasks: The first task is to compile our model, and the second task is to train the model with the preprocessed data produced from the Data class.
Detection	This class performs the intrusion detection task and generates the evaluation metrics.

5.3.1. IDS Class

The IDS class is the parent class which encapsulates rest of the four functioning classes and respective methods. It has two methods: *build()* and *execute()*. The *build()* method is responsible for instantiating all the other four classes with their respective methods. The *execute()* method just reuses the previously instantiated classes with a separate set of arguments. The two green branches in Figure 5.2 (above) represents the reuse functionality and the *param* represents the arguments. In the first green line *param=test-set*, which means the Data class is re-used by providing *test-set* as new argument. Similarly, the second green line (where *param=X_test*) signifies that the Detection class is re-used with *X_test* as argument. *X_test* has been discussed in detail in Section 5.3.2.1.3.

5.3.2. Data Class

The Data class deals with the data preprocessing mechanism and consists of only one method called *preprocess()*.The *preprocess()* method deals with data preprocessing and consists of a four sub-functions, including *importDataset(), normalizeData(), dataStructure()* and *reshape().* Details of the *preprocess()* method and its sub-functions are described in the following sections.

5.3.2.1. preprocess Method

The preprocess method takes the dataset (as a .csv file) as input parameter and reconstructs the data samples into a TensorFlow neural network compatible structure that can be used for training and testing purpose. The *preprocess()* method is being called twice: in the first time, it is called by the *build()* method in the IDS class for processing the training-set and in the second time by the *execute()* method in the IDS class for processing the test-set. In the first instance, the parameter passed to the *preprocess()* method is the 'UNSW_NB15_training-set_5000.csv' file which contains the training dataset, and in the second instance the 'UNSW_NB15_testing-set.csv' file, which contains the testing dataset. The *preprocess()* method encapsulates 4 sub-functions for performing the data pre-processing task. The processed files are returned to the main IDS classes and are used in the training and testing phases, respectively. The details of the sub-functions of the *preprocess()* method are presented in the following sections.

5.3.2.1.1. importDataset Method

This method imports the training-set for training the neural network. The training-set is imported as *Pandas* data-frame (a 2-dimensional labeled data structure). This is because, training a neural network requires a *NumPy* array format; and to generate a *NumPy* array out of the .csv file format, *Pandas* data-frame is essential. The *read_csv* method from *Pandas* library is used in order to import the training-set as data-frame format. Figure 5.3 below is a snapshot of the Spyder IDE 'Variable Explorer' showing the structure of the data-frame format. While importing data it's important to note that, it is necessary not only to select the exact columns (which are *service, sbytes, sttl, smean, ct_dst_sport_ltm*), but also to convert them into an array of numbers, because only numbers can be the input of neural networks. The data-frame is named as *dataset_train*. Below is the line of python statements which import the training set from a .csv file and convert it to a data-frame.

import Pandas as pd

dataset_train = pd.read_csv('UNSW_NB15_training-set_5000.csv') I

In (I), first the *Pandas* library class is imported with *pd* as its object. Then *pd.read_csv* (where *pd* is an object of the *Pandas* class and *read_csv* is an inbuilt method from *Pandas* class) is employed to read the training set and import the values as data frame and store them in *dataset_train*.

dataset_train - DataFrame

Index	id	dur	proto	service	state	spkts
0	1	1.1e-05	udp	-	INT	2
1	2	8e-06	udp	-	INT	2
2	3	5e-06	udp	-	INT	2
3	4	6e-06	udp	-	INT	2
4	5	1e-05	udp	-	INT	2
5	6	3e-06	udp	-	INT	2
6	7	6e-06	udp	-	INT	2
7	8	2.8e-05	udp	-	INT	2
8	9	0	arp	-	INT	1
9	10	0	arp	-	INT	1
10	11	0	arp	-	INT	1

Figure 5.3: A view of the training-set as data-frame format taken from Spyder IDE

Next, the columns from the training-set are selected and stored as a *NumPy* array by (II):

training_set = dataset_train.iloc[:, [3,7,10,27,35]].values II

In (II), a new *NumPy* array variable called *training_set* is introduced. This *training_set* variable contains the training samples from the dataset for training our proposed BLSTM RNN. To obtain the relevant training samples, relevant columns

need to be selected from the dataset. To do that, we have used the *iloc()* method to get the right index of the columns we want. The data-frame (*dataset_train*) and the *iloc()* method is used to specify the column numbers that we want. Essentially, the columns pointed to by these column numbers contain the features that we intend to select. The *UNSW_NB15_training-set_5451.csv* file (i.e. the training set) consists of 45 features in total (i.e. 44 columns, since index number of the *Pandas* dataframe starts from zero, hence 45 features is projected as 0 to 44 columns in the dataframe). For this research purpose, we have selected the feature sub-set proposed by [9] (discussed in Section 3.13). These 5 features are located in the 4^{th}, 8^{th}, 11^{th}, 28^{th} and 36^{th} columns in the original dataset. The index of the *iloc()* method (3,7,10,27,35) are actually the dataframe column index of those features. As we have discussed before that the index number of the dataframe starts from zero, hence column 1 in the original dataset is index 0 in *iloc()* method, column 4 is index 3, column 8 is index 7, so on and so forth. Since we have 5 features and 5000 training samples, hence, the *training_set* consists of 5 columns and 5000 rows where each row corresponds to a sample.

5.3.2.1.2. normalizeData Method

This sub-function deals with the data normalization. The input of this sub-function is the output of the *importDataset()* method (*dataset_train*). Normalization refers to rescaling real numbers by the use of the formulae in (5.1)

$$X_{nor} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{5.1}$$

In (5.1), X refers to the value of each data sample. X_{nor} is the normalized value, min(X) is the minimum of all the values in the training-set and max(X) is maximum of all the values in the training-set. To perform the data normalization, we introduce a new variable called *training_set_scaled* which will store the new normalized values, because it is recommended to keep the original non-normalized training-set separate from the normalized one. The normalizeData sub-function returns a normalized 2D array (called *training_set_scaled*) of real numbers in the range of 0 and 1. The normalization process is carried out by the following Python statements in (III).

```
from sklearn.preprocessing import MinMaxScaler

sclr = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sclr.fit_transform(training_set)
```

III

In (III), the *MinMaxScalar* class (imported from the *sklearn.preprocessing* class) is responsible for data normalization. We have implemented data normalization by creating an object of the *MinMaxScalar* class called *sclr*, where a feature range between 0 and 1 is defined to specify the minimum and maximum range of normalized values. In other words, it means that the values of the normalized training-set should be within the range of 0 to 1. The *fit_transform()* method of the *MinMaxScalar* class is used to perform the data normalization process. The original un-normalized training-set (*training_set*) is passed as a function argument. Since heterogeneous data-type is not supported by Python, the non-numeric data entries (also called categorical data) are converted to numeric values. The process of converting categorical data into numeric values is called Encoding. For instance, the dataset features such as '*service*' highlighted (yellow) in Table 5.3, is an example of categorical data. Table 5.4 shows the corresponding encoded values of the categorical data. Last four columns of Table 5.4 shows the normalized values. The *fit_transform()* method performs the encoding operation and converts the categorical data into numeric values. Both the normalization and encoding process are performed within this *normalizeData()* sub-function.

Table 5.3: Example of categorical data (marked yellow) and other un-normalized data

service	sbytes	sttl	smean	ct_dst_sport_ltm
-	530	254	53	1
ftp	7954	254	568	1
http	794	254	397	1
smtp	2516	254	252	1
-	816	62	82	1
ftp	534	254	53	1

Table 5.4: Categorical to numeric conversion (marked yellow) other normalized data

service	sbytes	sttl	smean	ct_dst_sport_ltm
0	4.97482e-05	1	0.144928	0
0.3	0.000189706	1	0.581781	0
0.5	0.000112984	1	0.342305	0
0.8	9.4411e-05	1	0.284334	0
0	0.000229947	1	0.707384	0
0.3	8.1587e-05	1	0.244306	0

5.3.2.1.3. dataStructure Method

In order to feed data into an RNN, a TensorFlow data structure is required for storing the features and labels. To transform the dataset into a TensorFlow data structure, two separate entities are created. The first entity will be *X_train,* which is the input of the RNN and then the second entity will be *Y_train,* which will contain the expected output of the RNN. So, technically, *X_train* will contain the prior observations (from time t-1 till time t), and *Y_train* will contain the expected observation at time t+1. Important to note here is that the RNN neuron takes the *X_train*, learns the correlations between the observations in the data samples, and generates a prediction out of the learning. This generated prediction is the actual output of the RNN neuron at time t. To calculate the efficiency of the neurons, this actual output is compared with *Y_train* containing the expected output. The data structure is created by (IV).

```
X_train = []
Y_train  = []
for i in range(1, 5451):
        X_train.append(training_set_scaled[i-1:i, 0])
        Y_train.append(training_set_scaled[i, 0])
X_train, Y_train  = np.array(X_train), np.array(y_train)
```

IV

In (IV), two newly introduced variables *X_train* and *Y_train are* initialized as empty lists. Then, these two entities *X_train* and *Y_train* are populated with the traffic observations from our training-set by using a for-loop (where *i* represents the time t)

ranging from 1 to the last index of our training-set i.e. 5451. The *X_train* is appended with observations ranging from time i-1 to time i by using the *append()* function. *Y_train* is similarly appended with the observation at time t+1. Since both *X_train* and *Y_train* are lists, converting them to *NumPy* arrays is crucial, so that they can be accepted by our BLSTM RNN model. This conversion is implemented by using *np.array()* function (where np is an object of *NumPy* class).

To summarize, the *dataStructure()* method builds the compatible data structure required to train and test the RNN model. This method takes training-set as an argument to generate *X_train* and *Y_train*, and test-set as argument for generating *X_test* and *Y_test*. Generation of *X_test* and *Y_test* are implemented through (V).

```
X_test = []
Y_test = []
for i in range(1, 4205):                                    V
    X_test.append(test_set_scaled[i-1:i, 0])
    Y_test.append(test_set_scaled[i, 0])
X_test, Y_test = np.array(X_test), np.array(Y_test)
```

X_test and *Y_test* are just similar to *X_train* and *Y_train*, except, they hold the test-set observations instead of training-set observations. The *dataStructure()* method takes the normalized 2D array, i.e., output of the *normalizeData()* method as input and returns *NumPy* arrays: *X_train* and *Y_train* (for training-set) and *X_test* and *Y_test* (for test-set).

5.3.2.1.4. reshape Method

To make the data structure compatible with the input format of our RNN, the *reshape()* method is used as shown below in (VI).

```
X_train = np.reshape(X_train, (5451,5, 1))                  VI
```

In (VI), the *np.reshape()* method from *NumPy* class is used to implement the reshaping of the *X_train*. As per Keras Recurrent Layer documentation[3], the input shape of an RNN should be a 3-D tensor with the following dimensions: batch-size, time-step, and input-dimension. The *np.reshape()* method actually generates a 3D

[3] (available: https://keras.io/layers/recurrent/)

tensor which is compatible for RNNs. The argument structure of the *np.reshape()* method is like this: *np.reshape (name of the array to be reshaped, (batch_size, time_step, input_dim))*. The first argument in (VI) is *X_train* because *X_train* is the array that needs to be reshaped. The batch size is 5450, followed by the time-step of 5. The time step is usually equals the column numbers of the input array. The input dimension is 1. The reusability feature of the object oriented programming has been exploited to implement the *reshape()* method. In order to reshape the training-set into an RNN-compatible input format, *X_train* is passed as an argument to this method along with the other corresponding parameter values. When *X_test* is passed as an argument to this method, it reshapes the *X_test* into a compatible format which is fit for testing the model.

Table 5.5 summarizes the input parameters and the final output of the *preprocess()* method. Data pre-processing is implemented through the *preprocess()* method. Since we have to perform the pre-processing for both the training-set and test-set, we have used the reusability feature of OOP to implement data pre-processing. To implement the pre-processing of the training-set, the *preprocess()* method is called (from inside the *build()* method) with the training-set as an argument. For implementing test-set pre-processing, the same *preprocess()* method is called again (from inside the *execute()* method) with the test-set as an argument. This is why, Table 5.5 and Table 5.6 are segmented into two: blue corresponds to training-set and green for test-set. The respective sub-functions of the *preprocess()* method like *importDataset()*, *normalizeData()*, *dataStructure()* and *reshape()* also work according to the input parameters of the *preprocess()* method. For example, when the input is training-set, the output of the *importDataset()* sub-function is *training_set*, and when the input is test-set, the output of the same sub-function *test_set*, so on and so forth. Precisely, the same sub-function yields different output in different implementation steps. Table 5.5 below lists all the sub-functions of the *preprocess()* method.

Table 5.5: Input and Output of the *preprocess()* method

Method name	Input parameters	Final outcome	Briefing
preprocess()	Training-set (*UNSW_NB15_training-set_5451.csv file)*	*X_train, Y_train*	This method pre-processes our training-set and test-set, which is a .csv format, into the compatible data format.
	Test-set (*UNSW_NB15_test-set.csv file*)	*X_test, Y_test*	

Table 5.6: I/O of the sub-functions of the *preprocess()* method

Sub-function Name	Input parameters	Output	Sub-function Briefing
importDataset()	*UNSW_NB15_training-set_5451.csv file*	*training_set* (*Pandas* dataframe object)	This sub-function converts the .csv format into *Pandas* dataframe format
	UNSW_NB15_test-set.csv file	*test_set* (*Pandas* dataframe object)	
normalizeData()	*training_set*	*training_set_scaled*	This sub-function performs the Normalization and scaling.
	test_set	*test_set_scaled*	
dataStructure()	Time-step, *training_set_scaled*	*X_train, Y_train*	This sub-function creates the data structure
	Time-step,	*X_test, Y_test*	

	test_set_scaled		compatible for RNN
reshape()	*X_train*, batch_size, time_step, input_dim	*X_train*	Essentially, a Tensorflow based neural network structure takes a 3D array as input. This sub-function takes the previously built data structure and reshapes it into a 3D array.
	X_test, batch_size, time_step, input_dim	*X_test*	

5.3.3. Classifier Class

This class implements the architecture of our proposed BLSTM RNN model. In order to make it more robust, dropout regularization have been utilized, which is a mechanism to prevent overfitting of the model. The model architecture is implemented through several steps. Table 5.7 lists all the required steps and their respective actions.

Table 5.7: List of steps and their corresponding actions for building the RNN

Steps	What it does?
Step 1	Import the Keras library and it's corresponding classes
Step 2	Initialize the neural network
Step 3	Add input layer and Dropout regularization
Step 4	Add hidden layers with corresponding neurons and add Dropout regularization
Step 5	Add the Output layer

Step1: Importing the Keras Library

The foremost step of implementation is concerned with importing the Keras library and packages by the following lines of Python code in (VII):

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers import Dropout
```

VII

In (VII), the *Sequential* class creates an RNN object representing a sequence of layers of the neural network The *Dense* class is used for generating the output layer of the model. The *Bidirectional* and *LSTM* class is used to generate the input layer and hidden layers. Lastly, the *Dropout* class is used to add some dropout regularization to the model.

Step2: Initialize the BLSTM RNN

In this step, the initialization of the neural network is implemented by (VIII):

```
classifier = Sequential()
```

VIII

In (VIII), the *Sequential* class from Keras initializes the neural network object called *classifier*. Here, we introduce a new name called '*classifier'*. This *classifier* is an object of the *Sequential* class which represents a sequence of RNN layers. Executing this line will initialize the RNN. In the implementation level, this identifier ('*classifier')* represents our RNN model that we are going to build. In other words, '*classifier'* will be the name of our proposed BLSTM RNN model.

Step3: Add Input layers and dropout regularizations

In this step, the input layer creation is implemented by (IX):

```
classifier.add(Bidirectional(LSTM(units = 5, activation = 'relu',
return_sequences=True), input_shape=(X_train.shape[1]))
```

IX

```
classifier.add(Dropout(0.2))
```

In (IX), the *add()* method (which is a built-in method of the *Sequential* class) generates the input layer. Inside the *add*() method, the *LSTM* class is used to add the LSTM units or neurons. Then the *Bidirectional* class is used as a wrapper class which wraps up the LSTM units and provide a BLSTM unit altogether.

As per Keras documentation, the *LSTM* class is to be provided with the following list-of-arguments[4]:

- *units = 5:* denotes the number of input neurons, which is 5 in our model.
- *activation = 'relu':* specifies the activation function we are using for our input layer, which is a ReLU (Rectified Linear Unit) function.
- *return_sequence = True:* specifies whether the values of the specific layer will be passed to the next layer or not. Since our proposed model is a fully connected BLSTM network, the value of this argument is set as *True*.
- *input_shape:* specifies the shape of the input array (*X_train*) that has been created previously in the Data class.

In the last line of (IX), the *add()* method of the *sequential* class is used for implementing the dropout regularization to our model with a dropout-rate of 0.2

Step 4: Add hidden layers with corresponding neurons and add Dropout regularization

In this step, 3 hidden layers are added to the network by (X):

```
# hidden layer 1
classifier.add(Bidirectional(LSTM(units = 220,
return_sequences=True)))
classifier.add(Dropout(0.2))

# hidden layer 2
classifier.add(Bidirectional(LSTM(units = 240,
return_sequences=True)))
classifier.add(Dropout(0.2))

# hidden layer 3
classifier.add(Bidirectional(LSTM(units = 260, ,
return_sequences=True)))
```

X

[4] https://keras.io/layers/recurrent/#lstm

classifier.add(Dropout(0.2))

In (X), the 1st, 2nd, and the 3rd hidden layers are comprised of 220, 240 and 260 neurons, respectively. The *return_sequence* and dropout regularization works the same way as explained before in the third step.

Step 5: Add the Output layer

In this step, the addition of the output layer is implemented by (XI):

classifier.add(Dense(units = 1, activation = 'sigmoid')) XI

In (XI), the *add()* method from *sequential* class implements the output layer generation. Since the output layer is fully connected to the previous BLSTM layer, hence the *Dense* class of the Keras library is used to implement the full connectivity. As the network will perform a binary classification, so, the *units* parameter is set to 1, which means only 1 neuron will be there in the output. The second parameter (*activation = 'sigmoid'*) implements the output layer activation function.

5.3.4. FitModel Class

This class implements the compilation and the NN training. Compiling is the conversion procedure of the high level source code to the machine level binary code. The compilation of the model is implemented through (XII):

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy') XII

In (XII), the *compile()* method of the *Sequential* class implements the compilation procedure. The *compile()* method takes two arguments: *optimizer* and the *loss* function. ADAM is employed as the network optimizer and 'binary_crossentropy' as the loss function. After compilation, the model training is implemented by (XIII):

regressor.fit(X_train, y_train, epochs = 100, batch_size = 132) XIII

In (XIII), the RNN is trained with the training set. The *fit()* method is used to implement the training procedure. The *fit()* method connects the neural network to the training-set and perform iterations based on given parameters. The *fit()* method takes the following 4 arguments:

X_train: the input of the training set that has the features.
Y_train: the ground truth (i.e. the expected output) of the training set.
Epochs: is the number of epochs, i.e. number of iterations our neural network will be trained. In other words, it is the number of times the whole dataset will be propagated through the model. The implemented model will be trained using the same dataset for 100 times.
batch_size: finally, the batch_size is the amount of samples that are processed by the neural network at a time. The batch-size is 132, which means that the weights and biases of our network will get updated every 132 data samples.

5.3.5. Detection Class

The intrusion detection task and the evaluation metrics generation is implemented through the line of Python code in (XIV):

y_pred = classifier.predict(X_test) XIV

In (XIV), as a part of the implantation, we introduce a new variable called *y_pred*, which is a 1-D array that stores the computation values performed by the network. The method *predict()* is a method provided by Keras library which performs the computation process of intrusion detection. The *predict()* method takes only one argument, that is the pre-processed test-set (*X_test*). The generation of the *X_test* is been implemented by invoking the Data class from the *execute()* method with test-set ('UNSW_NB15_testing-set.csv' file) as parameter. The values returned by the *predict()* method is stored into the newly introduced variable *y_pred*.

Once *y_pred* is produced, the confusion matrix (*cm*) along with the classification report (*report*) generation is implemented through (XV). The classification report provides the evaluation metrics including precision, recall and f1-score, which are further discussed in Section 4.4 of Chapter 4.

from sklearn.metrics import confusion_matrix

```
from sklearn.metrics import classification_report
print( confusion_matrix(Y_test, y_pred))
print( classification_report(Y_test, y_pred))
```
XV

In (XV), the Python codes implement the evaluation phase. The *confusion_matrix()* and *classification_report()* method are imported from *sklearn* class. The *confusion_matrix()* method takes two arguments: *Y_test* and *y_pred*. The *Y_test* contains the expected output generated out of the test-set, and *y_pred* contains the actual output generated by our model. The *confusion_matrix()* method plots a graphical representation of the classification outcome by using these two arguments. The *classification_report()* also takes the same parameters and generates a tabular result consisting of precision, recall, f-1 score and false alarm rate.

5.4. Conclusions

In this chapter, we discussed the detailed step-by-step explanations of the implementation of our proposed BLSTM RNN model. We have followed an object oriented approach in order to perform the implementation of our proposed model. In order to facilitate the whole implementation, five classes is being constructed, namely: IDS class, Data class, Classifier class, FitModel class and Detection class. The IDS class is the parent class which instantiates the rest of the four classes and their corresponding methods. The Data class implements the pre-processing of the data-sets. The *preprocess()* method of the Data class implements the data pre-processing mechanism by taking the data-sets as input and producing the pre-processed RNN compatible data-structure as output. The Classifier class implements the proposed BLSTM RNN architecture by generating all the involved layers (1 input, 3 hidden and 1 output) along with the dropout regularizations. The training of the model is implemented through the FitModel class. The last class of our implementation scheme is the Detection class. This class implements the testing of the proposed model. This class takes test-set as input parameter and generates the confusion matrix and classification report as its output.

The *Keras* library have been used to implement the neural network and perform simulations. The *Keras* executes on top of Google TensorFlow which forms the backend framework of the implementation. The *Pandas* library is used to convert the

.csv format data-sets into RNN compatible data-frames. *NumPy* is a package in Python used for scientific computing. In order to manipulate arrays of unlike shapes (such as 2D array and 3D array), *NumPy* package is employed. The inbuilt scientific Python library called *sklearn* is used to implement the generation of evaluation parameters like confusion matrix and classification report.

Chapter 6

SIMULATION RESULTS & EVALUATION

This chapter shows the simulation results obtained as we adjust the model parameters like learning rate, batch size, time steps and dropout regularization and provide a qualitative performance analysis of the model. The model performance evaluation over different test-sets is also discussed.

6.1. Metric Definition and Clarification

The intrusion detection (ID) process is a binary classification problem where the system implementing the ID process classifies individual sample either “attack” or “normal”. The evaluation metrics are used to analyze the model performance, and the metrics include accuracy, recall, precision, f-1 score, false alarm rate (FAR) and miscalculation rate (error rate). Recall, is defined in (7.1).

$$recall = \frac{TP}{TP + FN} \tag{6.1}$$

Where, TP denotes the count of true positive samples (i.e. instances that are intrusions and are labeled by the model as intrusions) and FN denotes the amount of false negatives (i.e. instances that are intrusions but are labeled by the model as non-intrusions). Recall states the model capability of detecting all the “attack” samples within the dataset. It gives a sense of how good our model is in detecting “attack” samples from within the dataset. Precision is defined in (7.2).

$$precision = \frac{TP}{TP + FP} \tag{6.2}$$

Where, FP denotes the false positives (i.e. instances that are non-intrusions but are labeled by the model as intrusions). Precision states the ability of the model to identify only the relevant instances. It gives us a sense for how likely when the model labels a sample as positive, remains accurate.

The F1 score, defined in (7.3) is the harmonic mean of precision and recall taking both metrics into account.

$$f1-score=\frac{2(recall*\ precision)}{recall+\ precision} \quad 6.3$$

The overall accuracy of the model is calculated by (7.4).

$$Accuracy=\frac{TP+TN}{X} \quad 6.4$$

Where, X denotes total number of samples fed as input. TP is the true positives and TN denotes the true negatives (that is, instances that are normal and are labeled by the model as normal). Accuracy tells how often the model is correct.

The misclassification rate of the model defined in (7.5) signifies how often the classifier is wrong. Misclassification rate is the percentage of wrong detections and can be calculated by using the formulae in (4.2):

$$miscalculation\_rate=\frac{FP+FN}{X} \quad 6.5$$

False Positive Rate (FPR), also called False Alarm Rate (FAR) is calculated by (7.6).

$$FPR=\frac{FP}{X_{Normal}} \quad 6.6$$

Where, X_{Normal} is the number of actual normal samples in X. FPR or FAR is the percentage at which the model incorrectly classifies normal samples as intrusions.

6.2. Performance over Different Hyper-Parameters

This section provides the investigation details of the model performance over different hyper-parameters. As described in Chapter 5, the proposed model consisted of one input layer with 5 neurons, three hidden layers with 220, 240, and 260 neurons respectively, and one output layer with one neuron. All the layers are densely interconnected with each other. Sigmoid was used as activation function. As an initial experiment, 5450 samples were considered from the UNSW-NB15 training dataset

and the outcome for 100 iterations were recorded. It is important to note here: in order to investigate the model performance in relation with different hyper-parameters, only one hyper-parameter at a time is applicable to change. For instance, while studying the model performance with respect to time-steps, rest of the hyper-parameters (batch-size, dropouts, learning rate) remains constant. In the following sections, the model performance over different hyper-parameters have been discussed.

6.2.1. Performance over Different Time-Steps

The model performance was studied with respect to varying time steps. Time step refers to the number of steps the RNN is unrolled in time. In other words, time step defines the memory capacity of an RNN cell. Table 6.1 shows the hyper-parameters which remains constant during different time-steps. Table 6.2 tabulates the results of different time steps ranging from 1 to 60.

Table 6.1: Constant hyper-parameter values (excluding time-steps)

Batch size	**Dropout**	**Learning rate**	**Epochs**
132	0.2	0.001	100

Table 6.2: Results of different time-steps

Time-Steps	**Accuracy**	**Precision**	**Recall**	**F1 score**
1	0.95	1	0.95	0.97
15	0.79	1	0.79	0.88
30	0.73	0.99	0.96	0.98
45	0.73	0.99	0.96	0.98
60	**0.98**	**0.99**	**1**	**1**

Table 6.2 above, shows that the model is at its best with a time step value of 60. A second interesting trend is the relatively poor performance of the model at 15, 30 and 45 time steps. Though there remains no major fluctuations in the precision, recall and f1 value, but the accuracy drops very sharp. This may have arisen due to the decreased number of examples present to the model causing it to require more than the allotted number of epochs to converge.

6.2.2. Performance over Different Batch-Size

The time step value was chosen as 60, as it performed best with respect to all the evaluation matrices. A batch-size dictates the amount of samples fed to the network at a point of time. Determining optimal batch size requires cross validation, so, started with a very large batch size of 1090 (by keeping in mind that the total sample size must be divisible by batch size. This is a convenient convention, though it's not mandatory). Table 6.3 shows the hyper-parameters which remains constant. Table 6.4 tabulates the results of different batch sizes.

Table 6.3: Constant hyper-parameter values (excluding batch-size)

Time steps	Dropout	Learning rate	Epochs
60	0.2	0.001	100

Table 6.4: Results of different batch-size

Batch size	Accuracy	Precision	Recall	F1 score
1090	0.92	0.99	0.94	0.97
545	0.92	0.99	0.94	0.97
220	0.92	0.99	0.94	0.97
132	**0.98**	**0.99**	**1**	**1**

It could be observed that when using a larger batch there is a degradation in the quality of the model. This is probably because the large batch size have a tendency to converge to sharp minima which leads to degraded generalization [120]. In contrast, small batch size shows a promising performance for our model with 100% f1 and recall value.

6.2.3. Performance over Different Dropout Rates

As per the previous outcomes, batch-size = 132 and time-steps = 60 performed optimal. In practice, RNNs can easily overfit the training data which may result in degrading the model performance. Dropout is a regularization technique explained in section 3.9, which is employed while network training in order to avoid network overfitting and improving the performance of the model. Table 6.5 shows the hyper-

parameters which remains constant. Table 6.6 tabulates the results of different dropout rates.

Table 6.5: Constant hyper-parameter values (excluding dropout rate)

Time steps	Batch size	Learning rate	Epochs
60	132	0.001	100

Table 6.6: Results of different dropout rates

Dropout	Accuracy	Precision	Recall	F1 score
0.2	0.98	0.99	1	1
0.3	0.92	0.99	0.94	0.97
0.5	0.92	0.99	0.94	0.97
0.8	**0.98**	**0.99**	**1**	**1**

The idle value for drop out ranges from 0.2 to 0.8 depending on the model architecture and dataset size [121]. Too large dropout values may result the network to underfit and too small dropout might result in overfitting. Determining the optimal value or the “sweet spot” is a trial and error method. Table 6.6 shows that a drop of value of 0.2 and 0.8 functions most appropriate for yielding a robust performance.

6.3. Performance on Reduced Test-set

After studying the hyper-parameter tuning outcome in the previous sections, the hyper-parameter values with best results were considered. Table 6.7 tabulates the architecture of the proposed model with all the optimum parameter values.

Table 6.7: Architecture of our model with all the optimum parameter values

Input layer	1 (5 neurons)
Hidden Layers	3 (220,240,260 neurons respectively)
Output Layer	1 (1 neuron)
Activation function	Sigmoid
Batch-Size	132
Time-Steps	60
Dropout Rate	0.8
Learning Rate	0.001
Epochs	100

After training, the training-set becomes known data to the NN model. For observing the model's performance over an unknown set of data, we fed our model with a test-set. For initial testing, a reduced test-set with considerably less amount of data samples were prepared. The idea of preparing a reduced test-set was to see the models performance over unknown data relatively quick. For instance, if that model yields unsatisfactory performance during testing, it is easier and faster to retest the model with the reduced test-set, rather than retesting the model with full test-set. Another purpose of creating a reduced test-set is that, the full UNSW-NB15 test-set comprises of 9 types of attacks in total, out of which 5 types are often present in IoT attacks (Analysis, Backdoor, Denial-of-Service, Worms, and Reconnaissance). Hence, only these 5 types of attack samples along with the 'normal' samples were extracted to prepare the reduced test-set. The reduced test-set samples were extracted from UNSW_NB15_testing-set.csv file, and contains 4206 test samples. Table 6.8 shows the number of anomalies and normal samples used in the test-set. Table 6.9 summarizes the four parameter values in the confusion matrix: TP, FN, FP and FN (confusion matrix parameters are discussed previously in Chapter 4). Table 6.10 shows the experimental outcomes.

Table 6.8: Number of samples used for classification (reduced test-set)

Class	**Sample size**
Attack	4094
Normal	112

Table 6.9: Confusion matrix values (reduced test-set)

Parameter	Number of Samples
TP	4027
TN	1
FP	10
FN	166

Table 6.10: Classifier performance over reduced test-set

Performance Measure	Percentage
Accuracy	0.9571
Precision	0.99
Recall	1
f1 - score	1
Miscalculation rate	0.041
FAR	0
Detection Time (sec)	2.19

The model is capable of detecting attacks over the reduced test-set, with more than 95% accuracy, i.e. the model is successful of classifying more than 95% of the samples correctly. The model generates a precision value of 99%. That is, whenever the model labels a sample as “attack” or “normal”, it is 99% accurate. Our model generates a recall value of 100%, which indicates that the model is capable of detecting 100% of all the attack instances present in the data-set. The model generates a zero false alarm rates and a very low wrong detection rate of 4.1%. The proposed model was capable of classifying 4205 samples of data in 2.19 seconds on an Intel Core i7 2.4GHz Central Processing Unit (CPU) without a Graphics Processing Unit (GPU), which is impressively fast.

6.4. Performance on Full UNSW-NB15 Test-set

The full UNSW-NB15 testing-set comprises of 82332 samples and 9 attack types (namely: Analysis, Backdoor, Denial-of-Service, Worms, Reconnaissance, Shell code, Exploits, Fuzzers and Generic). In order to test the model's capability of detecting completely unknown attack types, the UNSW-NB15 testing-set was intentionally kept intact, which consisted of 4 new attack samples (Shell code, Exploits, Fuzzers and Generic). Table 6.11 shows the four parameter values in the confusion matrix. Table 6.12 shows the detailed report of the outcomes.

Table 6.11: Confusion matrix values over full UNSW-NB15 test-set

Parameter	**Number of Samples**
TP	79966
TN	21
FP	2102
FN	242

Table 6.12: Classifier performance over reduced full UNSW-NB15 test-set

Performance Measure	**Percentage**
Accuracy	0.9715
Precision	0.99
Recall	0.97
f1 - score	0.98
Miscalculation rate	0.028
FAR	0
Detection Time (sec)	36.2

From Table 6.12, it could be observed that our proposed model is capable of detecting attacks in the full UNSW-NB15 test-set with more than 97% accuracy. That is, the model is successful of detecting more than 97% of the attack and normal samples correctly. Interestingly, as the full test-set comprises of 4 new attack types,

the accuracy score shows that our model can also classify completely new attack types as well. An impressive precision value of 0.99 shows that whenever our model classifies a data sample as “attack” or “normal”, the model remains 99% correct in its classification. A satisfactory recall value of 0.97 indicates that the model is capable of detecting 97% of all the attack instances present in the full UNSW-NB15 test-set, including the new attack types. The model generates a false alarm rate of 0.02, indicating that the proposed model very seldom fires a false alarm to the user. A miscalculation rate or wrong detection rate signifies how often the model classification is wrong. A very low wrong detection rate of 0.02 signifies that our proposed model exhibit a very negligible detection error. The model exhibits impressive speed and was capable of classifying 82332 samples of data in only 36.2 seconds in an Intel Core i7 2.4GHz Central Processing Unit (CPU) without a Graphics Processing Unit (GPU).

Chapter 7

Conclusion & Future Work

Prime purpose of this research work was to detect intrusions in IoT network. To accomplish the objective, Artificial Neural Network (ANN), specifically, Bi-directional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN), a deep learning approach, and Googles ML framework termed TensorFlow was adopted and utilized through Python programing language. This research work shows that DL could effectively cope with securities in IoT network. The proposed model can detect five type of attacks that occur to IoT network, namely: Analysis, Backdoors, DoS, Reconnaissance and Worms.

IDS are evaluated by the attained accuracy of intrusion detection including the false alarm rate (FAR) of the model. The proposed IoT intrusion detection model demonstrated over 97% accuracy in effectively identifying attack and normal samples. The proposed model reported a FAR of 2.5%. This value is equivalent to the model's general misclassification rate, which constitutes a false negative rate. The proposed model's sensitivity is found to be 100% in (shown in section 6.3), which implies an impressive 0% false negative rate

The main contributions of this thesis are that, it investigates and explains the efficiency of DL algorithms in addressing intrusion detection in IoT systems. Secondly, it shows the efficiency of BLSTM RNN in detecting IoT attacks through simulation results and shows the parameter values essential for BLSTM RNN to generate high detection accuracy. This research work also contributes to the efficient way of implementing BLSTM RNN approach by using Python programming language and Google TensorFlow implementation framework.

This research work employs one of the most recent benchmark intrusion dataset called UNSW-NB15 which is a synthetic dataset restricted to only 5 types of attacks that occurs in any IoT network: Backdoor, DoS, Reconnaissance, Analysis and Worms. In future work, we are planning to enrich the IoT attack dataset by adding more IoT attack types with real IoT network traffic. The data pre-processing stage of the thesis was done manually which took a lot of working hours. This is because the source dataset was not in acceptable TensorFlow format. The recommendation to this drawback is that further work should be done to automate this process. The thesis work was analyzed only on CPU. The future recommendation is that the model should be analyzed on different computing resources like GPUs and should port the model to

different platforms such as iOS, Android, Google Clouds, CUDA etc. to test the performance of the proposed BLSTM RNN model. The best results generated by the algorithm depended on parameters such as batch size, epochs, learning rate, time steps. The values of these parameters were set manually per every iteration until the best results was achieved. For future development we will be working on automating the assigning of values for these parameters. This will ease the guess work and try and error approach of getting the best values that will produce the best detection accuracy.

References

[1] M. U. Farooq, M. Waseem, A. Khairi, & S. Mazhar, “A critical analysis on the security concerns of the internet of things (IoT)”, in *International Journal of Computer Applications*, *111, 2015.*

[2] Kevin Ashton, That Internet of things thing. Available: http://www.rfidjournal.com/articles/view?4986

[3] X. Yuan, C. Li and X. Li, “DeepDefense: Identifying DDoS Attack via Deep Learning,” in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, Hong Kong, 2017, pp. 1-8

[4] B. A. Tama and K. H. Rhee, “Attack Classification Analysis of IoT Network via Deep Learning Approach,” in *Research Briefs on Information & Communication Technology Evolution (ReBICTE)*, 2018. Doi: 10.22667/ReBiCTE.2017.11.15.015.

[5] Fu *et al*., “An Intrusion Detection Scheme Based on Anomaly Mining in Internet of Things”, in *IEEE International Conference on Wireless, Mobile & Multimedia Networks (ICWMMN 2011),* Beijing, 2011, pp. 315-320.
DOI: 10.1049/cp.2011.1014.

[6] Y. Bengio, A. Courville, and P. Vincent, “Representation Learning: A Review and New Perspectives,” no. 1993, pp. 1–30, 2012.

[7] G. Hinton, “Machine Learning Lecture 5: Distributed Representations Localist representations,” 2011.

[8] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research”, *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77, 2007.

[9] T. Janarthanan and S. Zargari, "Feature selection in UNSW-NB15 and KDDCUP'99 datasets", in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, Edinburgh, 2017, pp. 1881-1886. doi: 10.1109/ISIE.2017.8001537

[10] H. Kurniawan, Y. Rosmansyah & B. Dabarsyah, “Android anomaly detection

system using machine learning classification.", in *International Conference on Electrical Engineering and Informatics (ICEEI)*, 2015. DOI:10.1109/ICEEI.2015.7352512.

[11] Jung, E., Cho, I., & Kang, S. M., "An Agent Modeling for Overcoming the Heterogeneity in the IoT with Design Patterns", in *Park, J., Adeli, H., Park, N. and Woungang, I. (Eds.) Mobile, Ubiquitous, and Intelligent Computing*. Vo. 274, pp. 69-74.

[12] A. Javaid, Q. Niyaz,W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *9th EAI Int. Conf. Bio-inspired Inf. Commun. Technol. (BIONETICS),* New York, NY, USA, May 2016, pp. 21_26.

[13] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 258_263.

[14] M. Sheikhan, Z. Jadidi, and A. Farrokhi, "Intrusion detection using reduced-size RNN based on feature grouping," *Neural Comput. Appl.*, vol. 21, no. 6, pp. 1185_1190, Sep. 2012.

[15] X. Yuan, C. Li and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP),* Hong Kong, 2017, pp.1-8.

[16] Pfahringer Bernhard, "Winning the KDD99 classification cup: bagged boosting", ACM SIGKDD Explorations Newsletter,V.1, Issue 2, 2000, [Online] available: http://dl.acm.org/citation.cfm?id=846200

[17] Sabhnani M., and Serpen G., "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context", in *International Conference on Machine Learning, Models, Technologies and Applications,* pp. 209-215, 2003.

[18] Ghorbani A., Lu W., and Tavallaee M., 2010, "Network Intrusion Detection and Prevention: Concepts and Techniques", *Springer Science*, LLC.

[19] Kumar, G. Sunil, and C. V. K. Sirisha, “Robust Preprocessing and Random Forests Technique for Network Probe Anomaly Detection.,” *International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307*, Volume-1, Issue-6, January 2012. available: http://www.academia.edu/9521473/Robust_Preprocessing_and_Random_Forests_Technique_for_Network_Probe_Anomaly_Detection

[20] Bajaj and Arora, “Improving the Intrusion Detection using Discriminative Machine Learning Approach and Improve the Time Complexity by Data Mining Feature Selection Methods”, *International Journal of Computer Applications (0975-8887)*, Volume 76-No.1, August 2013. available: http://research.ijcaonline.org/volume76/number1/pxc3890587.pdf

[21] Pervez M. S. and Farid D. M., "Feature selection and intrusion classification in NSL-KDD cup 99 dataset employing SVMs," in *The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014),* Dhaka, 2014, pp. 1-6.

[22] Ingre B. and Yadav A., "Performance analysis of NSL-KDD dataset using ANN," in *International Conference on Signal Processing and Communication Engineering Systems,* Guntur, 2015, pp. 92-96.

[23] Moustafa N. and Slay J., 2015, “Unsw-nb15: A comprehensive data set for network intrusion detection,” in *MilCIS-IEEE Stream, Military Communications and Information Systems Conference*, Canberra, Australia, IEEE publication, 2015.

[24] Moustafa N. and Slay J., “The significant features of the UNSW-NB15 and the KDD99 sets for Network Intrusion Detection Systems”, in *the 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS 2015), collocated with RAID 2015,* 2016. Available: http://handle.unsw.edu.au/1959.4/unsworks_41254

[25] Aghdam Hosseinzadeh M. and Kabiri, “Feature Selection for Intrusion Detection System Using Ant Colony Optimization,” *International Journal of Network Security,* Vol 18, No.3, May 2016, pp.420-432. Available: http://ijns.jalaxy.com.tw/contents/ijns-v18-n3/ijns2016-v18-n3-p420-432.pdf

[26] G. Hinton, “Dropout : A Simple Way to Prevent Neural Networks from Overfitting,” vol. 15, pp. 1929–1958, 2014.

[27] Z. Dewa and L. A. Maglaras, “Data Mining and Intrusion Detection Systems,” vol. 7, no. 1, pp. 62–71, 2016

[28] T. Hayashi, S. Watanabe, T. Toda, T. Hori, J. Le Roux, and K. Takeda, “Bidirectional LSTM-HMM Hybrid System for Polyphonic Sound Event Detection”, *Mitsubishi Electric Research Laboratories (MERL),* 201 Broadway, Cambridge, MA 02139, USA,” no. September, pp. 2–6, 2016.

[29] C. Yin, Y. Zhu, J. Fei, and X. He, “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks,” vol. 5, 2017.

[30] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning," Nature, vol. 521, no. 7553, pp. 436_444, May 2015.

[31] V. Timčenko and S. Gajin, “Machine Learning based Network Anomaly Detection for IoT environments”.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *Nature*, 1986.

[33] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, *Science*, 313(5786):504–507, 2006.

[34] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, “Why does unsupervised pre-training help deep learning?”, *JMLR*, 11, 2010

[35] G. Li and Z. Yan, “Data Fusion for Network Intrusion Detection: A Review”, 2018.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *NIPS*, 2012.

[37] A Turing, “I.—computing machinery and intelligence”, *Mind*, LIX(236):433–460, 1950.

[38] T. M. Mitchell, “Machine Learning”, McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[39] M. Mohammadi, G. S. Member, A. Al-fuqaha, and S. Member, “Deep Learning for IoT Big Data and Streaming Analytics : A Survey,” pp. 1–34, 2017.

[40] A. Gulli and S. Pal, *Deep Learning with Keras*, April 2017. Birmingham: Packt Publishing Ltd.

[41] K. Panetta. (2016) Gartner’s top 10 strategic technology trends for 2017. [Online]. Available: http://www.gartner.com/smarterwithgartner/ gartners-top-10-technology-trends-2017/

[42] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[43] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673–2681, 1997.

[45] Alex Graves and Jurgen Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[46] H. Lee, “Framework and development of fault detection classification using iot device and cloud environment,” *Journal of Manufacturing Systems*, 2017.

[47] Mike Schuster and Kuldip K Paliwal, “Bidirectional recurrent neural networks,” *Signal Processing, IEEE Transactions*, vol. 45, no. 11, pp. 2673–2681, 1997.

[48] Z. Cui, S. Member, R. Ke, S. Member, and Y. Wang, “Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction,” pp. 1–12, 2018.

[49] A. Candel, V. Parmar, E. LeDell, and A. Arora, “Deep learning with h2o,” 2015.

[50] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, “Comparative study of deep learning software frameworks,” *arXiv preprint arXiv:1511.06435v3 [cs.LG],* 2016.

[51] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin et al., “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467v2 [cs.DC],* 2016.

[52] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[53] S. Raschka and V. Mirjalili, Python Machine Learning, 2nd ed. Birmingham, UK: Packt Publishing, 2017.

[54] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM international conference on Multimedia. ACM*, pp. 675–678, 2014

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[56] T. J. Hazen. (2016) Microsoft and liebherr collaborating on new generation of smart refrigerators. [Online]. Available: http: //blogs.technet.microsoft.com/machinelearning/2016/09/02/

[57] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, and C. Rieger, “Intelligent buildings of the future: Cyberaware, deep learning powered, and human interacting,” *IEEE Industrial Electronics Magazine*, vol. 10, no. 4, pp. 32–49, 2016.

[58] X. Song, H. Kanasugi, and R. Shibasaki, “Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level.” IJCAI, 2016.

[59] G. Mittal, K. B. Yagnik, M. Garg, and N. C. Krishnan, “Spotgarbage: smartphone app to detect garbage using deep learning,” in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing. ACM*, 2016, pp. 940–945.

[60] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, “Deep learning for decentralized parking lot occupancy detection,” *Expert Systems with Applications*, 2017.

[61] S. Valipour, M. Siam, E. Stroulia, and M. Jagersand, “Parking-stall vacancy indicator system, based on deep convolutional neural networks,” in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT),* 2016, pp. 655–660.

[62] A. Gensler, J. Henze, B. Sick, and N. Raabe, “Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks,” in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on. IEEE, 2016*, pp. 2858–2865.

[63] Y. Hada-Muranushi, T. Muranushi, A. Asai, D. Okanohara, R. Raymond, G. Watanabe, S. Nemoto, and K. Shibata, “A deep-learning approach for operation of an automated realtime flare forecast,” *SPACE WEATHER*, 2016.

[64] X. Ma, H. Yu, Y. Wang, and Y. Wang, “Large-scale transportation network congestion evolution prediction using deep learning theory,” *PloS one*, vol. 10, no. 3, p. e0119044, 2015.

[65] Y. Tian and L. Pan, “Predicting short-term traffic flow by long short-term memory recurrent neural network,” in *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on. IEEE*, 2015, pp. 153–158.

[66] M.-J. Kang and J.-W. Kang, “Intrusion detection system using deep neural network for in-vehicle network security,” *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[67] D. Cires¸an, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, 2012.

[68] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, and Y. Ma, “Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment,” in *International Conference on Smart Homes and Health Telematics. Springer*, 2016, pp. 37–48.

[69] C. R. Pereira, D. R. Pereira, J. P. Papa, G. H. Rosa, and X.-S. Yang, “Convolutional neural networks applied for parkinson’s disease identification,” in *Machine Learning for Health Informatics. Springer*, 2016, pp. 377–390.

[70] J. Wang, H. Ding, F. Azamian, B. Zhou, C. Iribarren, S. Molloi, and P. Baldi, “Detecting cardiovascular disease from mammograms with deep learning,” *IEEE Transactions on Medical Imaging*, 2017.

[71] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell, “Learning to diagnose with lstm recurrent neural networks,” in *ICLR 2016*, 2016.

[72] D. Rav`ı, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang, “Deep learning for health informatics,” *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 4–21, 2017.

[73] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, “Deep neural networks based recognition of plant diseases by leaf image classification,” *Computational Intelligence and Neuroscience*, vol. 2016, 2016.

[74] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov, “Deep learning classification of land cover and crop types using remote sensing data,” *IEEE Geoscience and Remote Sensing Letters, 2017*.

[75] K. Kuwata and R. Shibasaki, “Estimating crop yields with deep learning and remotely sensed data,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International. IEEE*, 2015, pp. 858– 861.

[76] G. J. Scott, M. R. England, W. A. Starms, R. A. Marcum, and C. H. Davis, “Training deep convolutional neural networks for land– cover classification of high-resolution imagery,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 4, pp. 549–553, 2017.

[77] K. A. Steen, P. Christiansen, H. Karstoft, and R. N. Jørgensen, “Using deep learning to challenge safety standard for highly autonomous machines in agriculture,” *Journal of Imaging*, vol. 2, no. 1, p. 6, 2016.

[78] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, “Deepfruits: A fruit detection system using deep neural networks,” *Sensors*, vol. 16, no. 8, p. 1222, 2016.

[79] H. Wang, N. Wang, and D.-Y. Yeung, “Collaborative deep learning for recommender systems,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM*, 2015, pp. 1235–1244.

[80] T.-Y. Yang, C. G. Brinton, C. Joe-Wong, and M. Chiang, “Behaviorbased grade prediction for moocs via time series neural networks,” *IEEE Journal of Selected Topics in Signal Processing,* 2017.

[81] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, “Deep knowledge tracing,” in *Advances in Neural Information Processing Systems*, 2015, pp. 505–513.

[82] F. Conti, A. Pullini, and L. Benini, “Brain-inspired classroom occupancy monitoring on a low-power mobile platform,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 610–615.

[83] A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster, "Deep learning in the automotive industry: Applications and tools," in *Big Data (Big Data), 2016 IEEE International Conference on. IEEE*, 2016, pp. 3759–3768.

[84] Q. Wang, Y. Guo, L. Yu, and P. Li, "Earthquake prediction based on spatio-temporal data mining: An lstm network approach," *IEEE Transactions on Emerging Topics in Computing,* 2017.

[85] Y. Liu, E. Racah, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, and W. Collins, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *Int'l Conf. on Advances in Big Data Analytics*, 2016.

[86] H. Maeda, Y. Sekimoto, and T. Seto, "Lightweight road manager: smartphone-based automatic determination of road damage status by deep neural network," in *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems. ACM*, 2016, pp. 37–45.

[87] W. Liu, J. Liu, X. Gu, K. Liu, X. Dai, and H. Ma, "Deep learning based intelligent basketball arena with energy image," *in International Conference on Multimedia Modeling. Springer*, 2017, pp. 601–613.

[89] K.-C. Wang and R. Zemel, "classifying nba offensive plays using neural networks," in *Proc. MIT SLOAN Sports Analytics Conf*, 2016.

[90] T. Kautz, B. H. Groh, J. Hannink, U. Jensen, H. Strubberg, and B. M. Eskofier, "Activity recognition in beach volleyball using a deep convolutional neural network," *Data Mining and Knowledge Discovery*, pp. 1–28, 2017.

[91] M. S. Ibrahim, S. Muralidharan, Z. Deng, A. Vahdat, and G. Mori, "A hierarchical deep temporal model for group activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1971–1980.

[92] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, “Theano: new features and speed improvements,” *arXiv preprint arXiv:1211.5590v1 [cs.SC],* 2012.

[93] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[94] S. Raschka and V. Mirjalili, Python Machine Learning, 2nd ed. Birmingham, UK: Packt Publishing, 2017.

[95] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM international conference on Multimedia. ACM,* 2014, pp. 675–678.

[96] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-theart deep learning software tools,” *arXiv preprint arXiv:1608.07249v7 [cs.DC]*, 2016.

[97] R. Mehmood, F. Alam, N. N. Albogami, I. Katib, A. Albeshri, and S. M. Altowaijri, “Utilearn: A personalised ubiquitous teaching and learning system for smart societies,” *IEEE Access*, vol. 5, pp. 2615– 2635, 2017.

[98] A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster, “Deep learning in the automotive industry: Applications and tools,” in *Big Data (Big Data), 2016 IEEE International Conference on. IEEE*, 2016, pp. 3759–3768.

[99] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, “Comparative study of deep learning software frameworks,” *arXiv preprint arXiv:1511.06435v3 [cs.LG],* 2016.

[100] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-theart deep learning software tools,” *arXiv preprint arXiv:1608.07249v7 [cs.DC]*, 2016.

[101] Z. Cui, S. Member, R. Ke, S. Member, and Y. Wang, “Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction,” 2016, pp. 1–12.

[102] P. Barham et al., “TensorFlow : A system for large-scale machine learning,” pp. 265–284.

[103] R. C. Staudemeyer, “Applying long short-term memory recurrent neural networks to intrusion detection,” no. 56, pp. 136–154, 2015.

[104] A. Senior, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling Has.”

[105] H Ning, “Unit and Ubiquitous Internet of Things”, *CRC Press Inc*.,2013.

[106] E. Jung *et al*., “An Agent Modeling for Overcoming the Heterogeneity in the IoT with Design Patterns.” in *Park, J., Adeli, H., Park, N. and Woungang, I. (Eds.) Mobile, Ubiquitous, and Intelligent Computing*, Vol. 274, pp. 69-74, 2014.

[107] IERC. (2014) Internet of Things. Available: http://www.internet-of-thingsresearch.eu/about_iot.htm.

[108] A. Oracevic, S. Dilek, and A. Oracevic, “Security in Internet of Things : A Survey Security in Internet of Things : A Survey,” 2017. doi: 978-1-5090-4260-9

[109] SHEN changxiang, ZHANG Huanguo and FENG Dengguo, “Literature Review of Information Security”, *Science in China (Series E: Information Sciences),* vol.37, no.2, 2007, pp.129-150

[110] M. A. Bhabad and P. G. Scholar, “Internet of Things : Architecture, Security Issues and Countermeasures,” vol. 125, no. 14, pp. 1–4, 2015.

[111] M. Weber, “Security challenges of the Internet of Things,” pp. 638–643, 2016.

[112] Shalev-Shwartz, S. & Ben-David, S., “Understanding Machine Learning: From Theory to Algorithms”, Cambridge University Press, 2014.

[113] K. Lackner, “Composing a melody with long-short term memory (LSTM) Recurrent Neural Networks,” 2016.

[114] Nerney, C. (2012) The tiny (yet powerful) world of speckled computing. Available: http://www.itworld.com/article/2721483/consumer-tech-science/the-tiny--yetpowerful--world-of-speckled-computing.html.

[115] “Introduction to Artificial Neural Networks - Part 1.” [Online]. Available: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7

[116] D. Williams and G. Hinton, “Learning Representations by Back-Propagating Errors,” *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[117] S. Ruder, “An Overview of Gradient Descent Optimization Algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[118] H. A. Abdul-ghani, D. Konstantas, and M. Mahyoub, “A Comprehensive IoT Attacks Survey based on a Building-blocked Reference Model”, vol. 9, no. 3, 2018.

[119] M. Abomhara and G. M. Køien, “Cyber Security and the Internet of Things : Vulnerabilities , Threats , Intruders,” vol. 4, pp. 65–88, 2015.

[120] G. E. G. Ap, S. H. M. Inima, J. Nocedal, P. Tak, and P. Tang, “o n l arge -b atch t raining for d eep l earning :,” pp. 1–16, 2017.

[121] Brownlee, J., “A Tour of Machine Learning Algorithms”. Available: http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/.

[122] N. Richárd. (2018, Sep 5). *The Big Difference between Artificial and Biological Neural Networks* [Online]. Available: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[123] A. Jonathan. (2016, Feb 21). *What is the unit Step Function in Artificial Neural Network?* [Blog]. Available: https://www.quora.com/What-is-the-unit-step-Function-in-Artificial-Neural-Network

Appendix - A

Conference

1. B. Roy and H. Cheung, “A Deep Learning Approach for Intrusion Detection in Internet of Things using Bi-Directional Long Short-Term Memory Recurrent Neural Network,” presented at *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, Sydney, Australia, 2018.

A Deep Learning Approach for Intrusion Detection in Internet of Things using Bi-Directional Long Short-Term Memory Recurrent Neural Network

Bipraneel Roy
School of Computing, Engineering and Mathematics
Western Sydney University
Sydney, Australia
B.Roy2@westernsydney.edu.au

Dr. Hon Cheung
School of Computing, Engineering and Mathematics
Western Sydney University
Sydney, Australia
H.Cheung@westernsydney.edu.au

Abstract— **Internet of Things (IoT) is one of the most rapidly evolving technologies nowadays. It has its impact in various industrial sectors including logistics tracking, medical fields, automobiles and smart cities. With its immense potentiality, IoT comes with crucial security concerns that need to be addressed. In this paper, we present a novel deep learning technique for detecting attacks within the IoT network using Bi-directional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN). A multi-layer Deep Learning Neural Network is trained using a novel benchmark data set: UNSW-NB15. This paper focuses on the binary classification of normal and attack patterns on the IoT network. The experimental outcomes show the efficiency of our proposed model with regard to precision, recall, f-1 score and FAR. Our proposed BLSTM model achieves over 95% accuracy in attack detection. The experimental outcome shows that BLSTM RNN is highly efficient for building high accuracy intrusion detection model and offers a novel research methodology.**

Keywords—Bi-directional Recurrent Neural Network, Deep Learning, Intrusion Detection, IoT.

I. INTRODUCTION

Internet of Things (IoT) was initially termed by Kevin Ashton in the year 1999 [2]. It stands for a system of globally recognizable physical devices or things which can sense the environment around them and behave intelligently. IoT is rising at an accelerating stride and interconnecting billions of devices or 'things'. As per Gartner, about 25 billion distinctively recognizable objects or things are predicted to be a part of the worldwide computing system by 2020 [1]. These interconnected devices augment regular activities and shape smart solutions. However, the immense prospects and conveniences brought by IoT lead to security concerns. IoT is considered as the future Internet or Internet 2.0. Consequently, IoT acquires traditional Internet security concerns as well as some new ones [18]. An IoT system can be a victim of several ways of attacks: Physical attacks (e.g. Node tampering, Node jamming, etc.), Network attacks (e.g. the Sinkhole attack, Denial-of-Service attacks, Man-in-the-Middle (MiM) attacks, etc.), Software attacks (e.g. Worms, Trojan horse, Spyware, etc.) and Encryption attacks (e.g. Cryptanalysis attacks) [19]. In this paper, we focus on detecting network attacks only, which target the network layer of IoT, and the attackers do not essentially have to be nearby the IoT system to perform such attacks [19].

An intrusion detection system (IDS) is a security system capable of scanning the network traffic activity and can identify any hostile or abnormal behavior. Technically, an IDS is equivalent to a classification task, i.e., identifying whether any network behavior is "abnormal" or "normal". Any classification problem can be of two types: binary classification and multi-class classification. In binary classification the system generates only one of two outputs: "attack" or "normal". A multi-class classification, on the other hand, identifies attack types as well. In this paper, we employ binary classification for intrusion detection.

Most of the conventional ML techniques use shallow learning and is incapable of effectively solving the intrusion classification problem with the immense data from a real-time environment [3]. In contrast, with the vibrant evolution of various datasets, deep learning (DL) approaches possess the prospective to mine or extract improved representations out of the data and can extract much more efficient features. The concept of deep learning was introduced by G. Hinton et al. [4] in the year 2006 and over the years, deep learning has undergone a spectacular rise in the area of ML. Since deep learning has the property of the automated discovery of abstraction from the raw data set, to build a much more efficient intrusion detection model, we propose a unique deep learning methodology to build an IDS for IoT using the Bi-directional Long Short-Term Memory Recurrent Neural Networks (BLSTM RNN) approach.

This paper presents a BLSTM RNN intrusion detection model and its implementation. The model's performance in binary classification is studied with respect to accuracy, miscalculation rate, precision, true positive rate and f-1 score. The intrusion detection model is implemented using the Python program language, Google TensorFlow, and Keras. Simulations will be performed using the UNSW-NB15 dataset. The experimental outcomes exhibit the efficiency of our proposed BLSTM RNN model in detecting 5 types of security attacks that an IoT network may encounter. The remaining of this paper is structured as follows. Section II comprises related work within the field of intrusion detection. Then, section III describes the introduced model for intrusion detection, including the benchmark UNSW-NB15 dataset, data pre-processing mechanism and evaluation matrix. Section IV highlights the experimental outcomes and discussions. Finally, Section V presents the conclusions and the future scope of this research.

II. RELATED WORK

A recent work by B. A. Tama and K. H. Rhee [5] proposes a deep neural network (DNN) model for attack classification in IoT network, where instead of employing previous data sets (NSL-KDD and KDD-99), the authors have evaluated the performance of their DNN model using three contemporary benchmark data sets: GPRS, CIDDS-001 and UNSW-NB15. Their study also reports an occurrence of bias results in CIDDS-001 dataset due to a data imbalance issue, that is the distribution of one class in CDDS-001 dataset is compellingly lower than the supplementary class. The study also remains unable to observe the performance differences between the DNN and other machine learning algorithms.

In recent years, deep learning has developed progressively, and has become functional for detecting intrusions and outperforming conventional methods. In [6], a deep learning method has been used by employing a DNN for flow-based anomaly recognition. The outcome reveals that the proposed technique could be used for detecting anomalies in software-defined systems. In [7], a deep learning technique has been proposed where the authors use a self-taught-learning (STL) algorithm on the NSL-KDD dataset. When relating the performance with former studies, the approach has proved to be more efficient. However, their studies emphasize only on the feature reduction capability of of DL techniques.

Fu et al. [8] introduces a novel intrusion detection technique intended for IoT systems established upon anomaly extraction. In their study, the authors assert that anomalies are detectable by analyzing the patterns of the data of the IoT sensor layer, like the temperature, humidity or anything that an IoT object sensor could collect and report. The study uses an unsupervised algorithm for data-mining for identifying normal patterns. In order to evaluate the proposed system, Intel Lab Project dataset was used, but no detected accuracy was reported to the designed system.

Another study conducted by M. Sheikhan et al. [9] claims that RNNs can be viewed as reduced-sized neural networks (NNs). The paper introduces a 3-layer RNN architecture having 41 input features and 4 intrusion classes as outputs for a misuse-based intrusion detection system. Nevertheless, the RNN units of layers remain partly connected. As a result, the proposed RNNs does not exhibit the capability of DL to produce high dimensional features. Moreover, performance evaluation of the proposed approach in terms of binary classification has not been reported.

With the consistent growth of big data along with the increase in computational power, the deep learning technique has become popular rapidly, and is increasingly utilized in numerous fields. In this paper, an unique deep learning technique has been proposed for detecting intrusions in the IoT network by using a bidirectional LSTM (BLSTM) recurrent neural network (RNN). Related with former works, we have used the BLSTM-based model aimed at binary classification and excluding pre-training. In addition, we have used two distinct data sets for training and testing purposes (namely, UNSW-NB15_training-set.csv and UNSW-NB15_test-set.csv) for evaluating the performance of the proposed model.

III. PROPOSED MODEL

A. Reccurrant Neural Network

A Recurrent Neural Network (RNN) is a layered network with feedback loops and is able to propagate past information onward to the present time. An RNN consists of loops and these loops allow the information to persist. The hidden layers of the RNN act as information storage like computer memory. RNNs form a class of powerful DNNs that use its internal memory along with loops for dealing with sequence data [20].

B. Long Short-Term Memory

Long Short-Term Memory (LSTM) is an extension of RNNs. LSTM employs the idea of gates for its units. One major issue with RNNs is that it is unable to learn the context information across a prolonged span of time caused by the vanishing gradient problem, which is, during a long temporal gap (i.e. time from when an input is obtained to the time when the input is used to make a prediction). Therefore, RNNs are incapable of learning from long-distance dependencies [21]. One solution to this issue is the use of an LSTM design [21]. It averts the issue of the vanishing gradient and thus permits the retention of the elongated period of context information.

C. Bi-directional LSTM

The concept of Bi-directional LSTM (BLSTM) originates from bidirectional RNN (BRNN) [10] that processes sequences of the input in forward as well as backward directions by employing two different hidden layers. Fig. 1 illustrates a bidirectional LSTM structure with three consecutive time steps.

BLSTMs join both the hidden layers to the same output layer. One inadequacy of traditional RNNs is that they are only capable of using the previous context of the input data sequence. BLSTMs [11] fix this by dispensing data in both forward and backward directions.

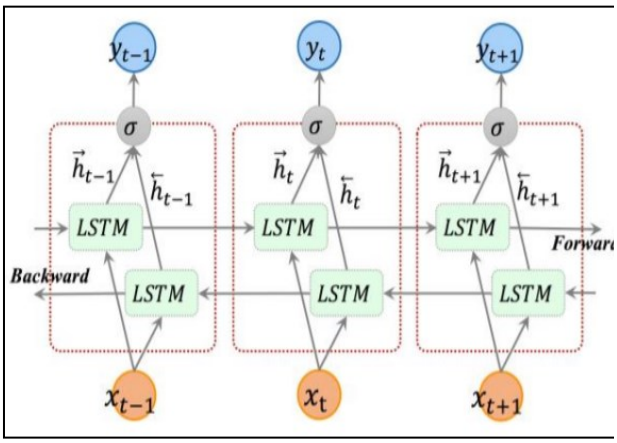


Figure 7.1: Bi-directional LSTM architecture with three consecutive time steps.

A BLSTM network computes the forward hidden layer sequence output $\vec{h}$, the output sequence of the backward hidden layer $\overleftarrow{h}$ and the output layer $\boldsymbol{y}$ by reiterating the forward layer starting *t =1 to T*, backward hidden layer since *t = T to 1*, and then the final output is upgraded by the following equations:

$$\vec{h}_t = H(W_{x\vec{h}}X_t + W_{\vec{h}\vec{h}}\vec{h}_{t+1} + b_{\vec{h}}) \quad (1)$$

$$\overleftarrow{h}_t = H(W_{x\overleftarrow{h}}X_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \quad (2)$$

$$Y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y) \quad (3)$$

The final output vector, $\boldsymbol{Y}_T$ is calculated by the equation:

$$Y_t = \sigma(\vec{h}_t, \overleftarrow{h}_t) \quad (4)$$

The σ function combines both the output sequences from the neurons in the hidden layers and can be one of four functions: concatenation, summation, averaging and multiplication.

Incorporating BRNNs with LSTM neurons results a bidirectional LSTM recurrent neural network (BLSTM RNN) [12]. The BLSTM RNN is capable of accessing long-term context data in both the backward and forward directions. The combination of both the forward and backward LSTM layers is considered as a single BLSTM layer. It has been shown that the bidirectional models are considerably better than regular unidirectional models in various domains like phoneme classification and speech recognition [13].

D. Intrusion Dataset

Moustafa and Slay (in 2015) [14] suggested that the NSL-KDD dataset and KDD'99 dataset did not characterize the up-to-date features for intrusion detection, and presented a comprehensive and all-inclusive dataset called the UNSWNB15. This dataset encompassed several features from KDD'99 dataset [15]. They further analyzed the features of the KDD'99 dataset and the UNSW-NB15 dataset. The results demonstrated that actual KDD'99 dataset features were less representative as compared to the features of UNSW-NB15 dataset [15].

The UNSW-NB15 dataset contains 45 features [16]. The dataset is further divided into training and testing datasets that contain all the current attack types. T. Janarthanan and S. Zargari [15] performed an extensive study on the UNSW-NB15 dataset for the purpose of extracting the most competent features and thus proposed a subset with features which dramatically increased the intrusion detection efficiency. The UNSW-NB15 dataset is the most recent and effective dataset published for intrusion detection research purposes. In this paper, the dataset subset in the file 'UNSW-NB15_training-set.csv' is used for training the proposed IDS model, while that in 'UNSW_NB15_test-set.csv' is used for testing the model. Both the dataset files can be obtained from: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

The data set file 'UNSW-NB15_training-set.csv'contains 175,341 records for training, while the test set file 'UNSW-NB15_testing-set.csv' contains 82,332 records. the UNSW-NB15 dataset has 9 attack types in total, out of which 5 types are often present in IoT attacks (Analysis, Backdoor, Denial-of-Service, Worms, and Reconnaissance) [28][29][30]. Hence, we extracted only these 5 types of attack samples along with the 'normal' samples to prepare our test-set.

In [22], the authors have used UNSW-NB15 dataset for conducting IoT research because unlike previous benchmark datasets, UNSW-NB15 exhibits contemporary attack patterns and modern normal traffic patterns. Moreover, since UNSW-NB15 has separate training-set and testing-set, data distribution remains different [22]. Again, in [26], the authors points out that: "It encompasses realistic normal traffic behavior and combines it with the synthesized up to date attack instances". [27] also points out that previous benchmark data sets like KDD'99 and NSL-KDD could not meet the current network security research needs as they does not comprehend the present-day network security circumstances and the latest attack features.

We choose UNSW-NB15 data set for our research as it covers modern attack patterns, consists of modern normal traffic patterns, and contains only two classes ('attack' and 'normal'). Since we are performing binary classification task, this class distribution facilitates our proposed approach. Secondly, UNSW-NB15 forms a comprehensive data set that presents 5 types of IoT attacks. The categories of attack classes are discussed below:

1) *Analysis*

These types of attacks are targeted at IoT system networks. The attacker first acquires related network information through packet sniffing or port scanning and then launches attacks on the targeted network [28].

2) *Backdoor*

With the advancement of IoT, several proposed IoT operating systems such as Contik and RTOS might encompass backdoor where it is possible to reprogram them for getting access to confidential data stored or transmitted on the IoT networks [29].

3) *Denial-of-Service*

Over the application layer, an IoT network can be compromised by Denial of Service (DoS) attacks or Distributed Denial of Service (DDoS) attacks; where, the service becomes unavailable to the authentic users, because the system becomes unavailable due to overwhelming number of requests resulting in resources and capacity overload [28].

4) *Worms*

Worms are malicious software that can be executed on the IoT Application layer that could harm IoT System devices. For instance, Stuxnet and Mirai have been developed to attack IoT objects [29].

5) *Reconnaissance*

It is an umbrella term of any illegitimate mapping and discovery of vulnerabilities in systems and services. For example, packet sniffing, port scanning and traffic analysis [30].

E. Data Preprocessing

As an initial experiment, reduced dataset samples are randomly selected from the whole training set and placed in a new .csv Microsoft Excel file titled "UNSW_NB15_training-set_5451.csv". In addition, we only consider the attributes proposed in [15], namely, service, sbytes, sstl, smean, and ct_dst_sport_ltm. The training dataset

is manually manipulated using the approach of Fu et al. [8], where, the authors has followed the approach of manually adding some abnormal samples in the dataset in order to make the dataset fit for their research purpose. The benefit of using the approach is that the input dataset would be competent for intrusion detection, which would fit the goal of the research. Moreover the approach helps in dealing with the problem of procuring labeled intrusion detection IoT datasets at a high cost. Here, we have followed the approach of manual manipulation and have extracted the features and attack types manually. Thus, our resulting dataset consists of 5 features and two class labels: "Attack" and "Normal". Table I shows the dataset structure. The first 5 columns represent the extracted features, the 6th column represents the attack category and the last column is the binary labeling. Value 0 resembles 'normal' and value 1 as 'attack'.

F. Evaluation Matrix

The confusion matrix is applied to characterize the accuracy of our proposed BLSTM RNN model during testing. The confusion matrix is a 2-dimensional matrix representing the correlation amongst the detected and actual values as shown in Fig. 2. True Positive (TP) specifies the count of anomalous or unusual samples that are accurately detected by the system. True negative (TN) signifies the amount of normal samples which are detected as normal by the system. False Positive (FP) represents the count of normal samples which are recognized as anomalies. False Negative (FN) refers to the amount of attack samples which have been classified as normal.

TABLE 7.1: DATASET STRUCTURE AFTER PRE-PROCESSING

service	sbytes	sttl	smean	ct_ dst_ sport_ ltm	attack_cat	label
smtp	37178	31	715	1	Normal	0
-	1280	254	64	1	Reconnaissance	1
-	1280	254	64	1	DoS	1
-	1280	254	64	1	Backdoor	1
	156	254	78	1	Analysis	1
http	1308	254	131	1	Worms	1

Accuracy defines the percentage of correct classifications and can be calculated by using the formula in (5):

$$accuracy = \frac{TP + TN}{X} \tag{5}$$

where, X denotes total count of samples.

Misclassification rate is the percentage of wrong detections and can be calculated by using the formulae in (6):

$$miscalculation\_rate = \frac{FP + FN}{X} \tag{6}$$

False Positive Rate (FPR), calculated by (7), is the percentage at which the system incorrectly classifies normal samples as anomaly:

$$FPR = \frac{FP}{X_{Normal}} \tag{7}$$

where, X_{Normal} is the number of actual normal samples in X.

Other parameters for evaluating the proposed model include precision, recall and f1-score values. Precision is calculated as the ratio of correct positive detections to the total actual positive detections, as shown in (8):

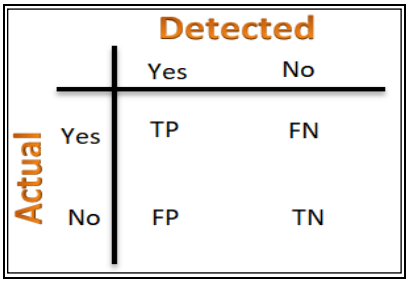
$$precision = \frac{TP}{TP + FP} \tag{8}$$

Recall is the ratio of correct positive detections to the number of actual abnormal samples, as presented in (9):

$$recall = \frac{TP}{TP + FN} \tag{9}$$

In (10), F1-Score denotes the harmonic mean of recall and precision:

$$f_1 - score = \frac{2(recall * precision)}{recall + precision} \tag{10}$$



These metrics are employed to assess the proposed intrusion detection model in the testing phase of the model simulations.

IV. IMPLEMENTATION

We have implemented the model in Spyder (a scientific interactive development environment for Python language) using Tensorflow library. The whole implementation process is divided into three major phases: data pre-processing, training the BLSTM model and lastly, testing and evaluation.

A. TensorFlow

In 2015 November, Google released an open source deep learning software library called TensorFlow [24]. The primary focus of TensorFlow is for defining, training and deploying machine learning algorithms. TensorFlow is a ML structure that functions at big scale and in diverse

environments. It employs dataflow graphs and maps the nodes or vertices of the graph across multiple machines incorporating graphics processing units (GPUs), multicore central processing units (CPUs) and Tensor processing units (TPUs). The architectural design provides a receptive and flexible platform for the application developers by allowing the developers to research with novel training algorithms and optimizations. TensorFlow supports several of applications, with an emphasis on training and implication on deep learning neural networks and it is being widely used for ML research [23]. Its supple dataflow representation aids power users to accomplish excellent performance.

B. Data Preprocessing phase

Data pre-processing forms the first phase of implementation stage. In this phase, the whole training dataset is read and stored in the computer memory. After that, feature extraction is employed. Since heterogeneous data type is not supported by Python, the non-numeric data entries (also called categorical data, such as the feature *service*, shown in table I) are then converted to numeric values. Dependent variables are encoded followed by data normalization (feature scaling). In order to process the features, we need to create a TensorFlow data structure for storing the features and labels. Since, we are employing an RNN, reshaping the data to respective time-steps is required. Reshaping forms the last step of data pre-processing phase.

C. Training phase

Training is the second phase of the implementation. First we have built the BLSTM RNN model by using Keras library. The model is then compiled and then followed by model-training. It is here, where the UNSW_NB15_training-set_5451.csv (reduced training-set) file is further divided into two subsets: Training set and Validation set, with a split ratio of 0.33%, i.e., 67% of the UNSW_NB15_training-set_5451.csv will be used for training, while 33% of for validating. The training subset is used by the compiler to train the model, while the validation subset is used for evaluating the model performance after each epoch.

After training the model, we analyse model's performance and repeat the training after tuning the model's parameters, until satisfactory performance is attained.

D. Testing phase

In this phase of our system, we load the test dataset and feed it into our trained model for the testing purpose. We then record the evaluation matrix for analysing our system.

V. RESULTS AND DISCUSSION

The Keras deep learning framework [17] and Google TensorFlow library are used to simulate the proposed BLSML RNN model. In the simulations, the proposed model basically performs binary classification where it classifies each input test sample as "normal" or "attack" in the testing phase. The evaluation metrics defined in the previous section, i.e., accuracy, error rate, precision, false positive rate, true positive rate, recall and F-1 score are used to evaluate the model performance in detecting intrusions. In the experiment, the simulated model was trained with a total of 5451 samples. The training samples were deduced from UNSW_NB15_training-set.csv file. The model was then tested with 4206 test samples. These test samples were extracted from UNSW_NB15_testing-set.csv file. Table II shows the number of anomalies and normal samples used in the test-set.

Table III summarizes the four performance values in the confusion matrix: TP, FN, FP and FN. Table IV shows the experimental outcomes. The proposed model is able to detect attacks using the reduced UNSW_NB15 dataset, with more than 95% accuracy with 100% precision. The model generates a zero false alarm rates and a very low wrong detection rate of 0.04% with an impressive recall and f1-score value of 98%. The proposed model was capable of classifying 4205 samples of data in 2.19 seconds on an Intel Core i7 2.4GHz Central Processing Unit (CPU) without a Graphics Processing Unit (GPU).

TABLE 7.2: NUMBER OF SAMPLES USED FOR CLASSIFICATION

Class	Sample size
Attack	4094
Normal	112

TABLE 7.3: SIMULATED RESULTS OF THE FOUR PERFORMANCE VALUES IN THE CONFUSION MATRIX

Parameter	Number of Samples
TP	4027
TN	1
FP	10
FN	166

TABLE 7.4: REPORTED ACCURACY, PRECISION, RECALL AND F1-SCORE OF THE PROPOSED CLASSIFIER INCLUDING MISCALCULATION RATE AND FAR

Performance Measure	Percentage
Accuracy	0.9571
Precision	1
Recall	0.96
f1 - score	0.98
Miscalculation rate	0.041
FAR	0
Detection Time (sec)	2.19

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented a new IDS model based on the BLSTM RNN for anomaly intrusion detection. The BLSTM RNN is able to perform deep learning effectively and to learn detailed features from the dataset in the training phase. This ability is important in learning characteristics in network traffic involved in anomaly intrusions to distinguish abnormal traffic from normal traffic.

We use Keras deep learning framework and Google TensorFlow library to implement the proposed new model. The implemented BLSTM was applied to a reduced dataset of the UNSW-NB15 dataset, which was used in several published works on IDS in IoT networks. The detection was based on binary classification, thus identifying normal and threat patterns. The developed model was able to achieve high accuracy in detecting attack traffic in the used dataset.

For future developments, more experiments will be performed to further analyse the proposed BLSTM RNN model using large data sets from published data sets, especially data sets containing dedicated IoT traffics. In addition, the developed model will be improved to increase

its detection accuracy further and the trade-offs between detection parameters.

VII. REFERENCES

M. U. Farooq, M. Waseem, A. Khairi, & S. Mazhar, “A critical analysis on the security concerns of the internet of things (IoT)”, International Journal of Computer Applications, 111, 2015.

Kevin Ashton, That Internet of things thing, It can be accessed at: http://www.rfidjournal.com/articles/view?4986.

C. Yin, Y. Zhu, J. Fei, and X. He, “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks,” vol. 5, 2017.

Y. LeCun, Y. Bengio, and G. Hinton, ``Deep learning," Nature, vol. 521, no. 7553, pp. 436_444, May 2015.

B. A. Tama and K. H. Rhee, “Attack Classification Analysis of IoT Network via Deep Learning Approach,” Research Briefs on Information & Communication Technology Evolution (ReBICTE), 2018. Doi: 10.22667/ReBiCTE.2017.11.15.015.

A. Javaid, Q. Niyaz,W. Sun, and M. Alam, ``A deep learning approach for network intrusion detection system," presented at the 9th EAI Int. Conf. Bio-inspired Inf. Commun. Technol. (BIONETICS), New York, NY, USA, May 2016, pp. 21-26.

T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, ``Deep learning approach for network intrusion detection in software defined networking," in Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM), Oct. 2016, pp. 258-263.

Fu *et al*., “An Intrusion Detection Scheme Based on Anomaly Mining in Internet of Things”, In IEEE International Conference on Wireless, Mobile & Multimedia Networks (ICWMMN 2011), Beijing, 2011, pp. 315-320. DOI: 10.1049/cp.2011.1014.

Bajaj and Arora, “Improving the Intrusion Detection using Discriminative Machine Learning Approach and Improve the Time Complexity by Data Mining Feature Selection Methods”, International Journal of Computer Applications (0975-8887), Volume 76-No.1, August 2013.

M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673–2681, 1997.

Mike Schuster and Kuldip K Paliwal, “Bidirectional recurrent neural networks,” Signal Processing, IEEE Transactions on, vol. 45, no. 11, pp. 2673–2681, 1997.

Alex Graves and Jurgen Schmidhuber, “Framewise ¨ phoneme classification with bidirectional LSTM and other neural network architectures,” Neural Networks, vol. 18, no. 5, pp. 602–610, 2005.

Z. Cui, S. Member, R. Ke, S. Member, and Y. Wang, “Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction,” pp. 1–12, 2018.

K. PEFFERS, T. TUUNANEN, M. ROTHENBERGER, and S. CHATTERJEE, A Design Science Research Methodology for Information Systems Research, Journal of Management Information Systems, vol. 24, no. 3, pp. 45-77, 2007.

T. Janarthanan and S. Zargari, "Feature selection in UNSW-NB15 and KDDCUP'99 datasets," *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, Edinburgh, 2017, pp. 1881-1886. doi: 10.1109/ISIE.2017.8001537

A. Javaid, Q. Niyaz,W. Sun, and M. Alam, ``A deep learning approach for network intrusion detection system," presented at the 9th EAI Int. Conf. Bio-inspired Inf. Commun. Technol. (BIONETICS), New York, NY, USA, May 2016, pp. 21-26.

Z. Dewa and L. A. Maglaras, “Data Mining and Intrusion Detection Systems,” vol. 7, no. 1, pp. 62–71, 2016.

M. Elkhodr, S. Shahrestani, and H. Cheung, “T HE INTERNET OF THINGS : NEW INTEROPERABILITY , MANAGEMENT AND SECURITY CHALLENGES,” vol. 8, no. 2, pp. 85–102, 2016.

M. Elkhodr, S. Shahrestani, and H. Cheung, “T HE INTERNET OF THINGS : NEW INTEROPERABILITY , MANAGEMENT AND SECURITY CHALLENGES,” vol. 8, no. 2, pp. 85–102, 2016.

Z. Cui, S. Member, R. Ke, S. Member, and Y. Wang, “Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction,” pp. 1–12, 2018.

T. Hayashi, S. Watanabe, T. Toda, T. Hori, J. Le Roux, and K. Takeda, “BIDIRECTIONAL LSTM-HMM HYBRID SYSTEM FOR POLYPHONIC SOUND EVENT DETECTION Mitsubishi Electric

Research Laboratories (MERL), 201 Broadway , Cambridge , MA 02139 , USA,” no. September, pp. 2–6, 2016.

B. A. Tama and K. H. Rhee, “Attack Classification Analysis of IoT Network via Deep Learning Approach,” Research Briefs on Information & Communication Technology Evolution (ReBICTE), 2018. Doi: 10.22667/ReBiCTE.2017.11.15.015.

P. Barham et al., “TensorFlow : A system for large-scale machine learning,” pp. 265–284.

N. Buduma, TensorFlow for deep learning—implementing neural networks. USA: O'Reilly Media, Inc., 2016.

V. Timčenko and S. Gajin, “Machine Learning based Network Anomaly Detection for IoT environments.”

G. Li and Z. Yan, “Data Fusion for Network Intrusion Detection : A Review,” 2018.

H. A. Abdul-ghani, D. Konstantas, and M. Mahyoub, “A Comprehensive IoT Attacks Survey based on a Building-blocked Reference Model,” vol. 9, no. 3, 2018.

M. Abomhara and G. M. Køien, “Cyber Security and the Internet of Things : Vulnerabilities , Threats , Intruders,” vol. 4, pp. 65–88, 2015.