LOW COMPLEXITY EFFICIENT ONLINE LEARNING ALGORITHMS USING LSTM NETWORKS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Ali Hassan Mirza

December 2018

Low Complexity Efficient Online Learning Algorithms Using LSTM Networks
By Ali Hassan Mirza
December 2018

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Tolga Çukur

Ramazan Gökberk Cinbiş

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

LOW COMPLEXITY EFFICIENT ONLINE LEARNING ALGORITHMS USING LSTM NETWORKS

Ali Hassan Mirza
M.S. in Electrical And Electronics Engineering
Advisor: Süleyman Serdar Kozat
December 2018

In this thesis, we implement efficient online learning algorithms using the Long Short Term Memory (LSTM) networks with low time and computational complexity. In Chapter 2, we investigate efficient covariance information-based online learning using the LSTM networks known as Co-LSTM networks. We utilize the covariance information into the LSTM gating structure and propose various efficient models. We reduce the computational complexity by applying the Weight Matrix Factorization (WMF) trick and derive the additive gradient based updates. In Chapter 3, we give a practical application of the network intrusion detection using the Co-LSTM networks. In Chapter 4, we propose a boosted binary version of Tree-LSTM networks which we call BBT-LSTM networks. We introduce the depth and windowing factor into the N-ary Tree-LSTM networks where each LSTM node is binarily split and the whole tree architecture grows in a balanced manner. In order to reduce the computational complexity of the BBT-LSTM networks, we apply WMF trick, replace the regular multiplication operator with the energy efficient operator and finally introduce the slicing operation on the BBT-LSTM network weight matrices. In Chapter 5, we propose another low complexity LSTM network based on a minimum number of hopping over the input data sequence. We study two methods to select the appropriate value of the hopping distance. Through an extensive set of experiments using the real-life data sets, we demonstrate the significant increase in the performance of the proposed algorithms at the end of each chapter.

Keywords: Online learning, LSTM, covariance, Tree-LSTM, boosting, regression, WMF.

ÖZET

UKSB AĞLARI İLE DÜŞÜK KARMAŞIKLIĞA SAHİP VERİMLİ ÇEVRİMİÇİ ÖĞRENME ALGORİTMALARI

Ali Hassan Mirza
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Aralık 2018

Bu tezde uzun-kısa soluklu bellek (UKSB) ağları ile verimli çevrimiçi öğrenme algoritmalarını takdim etmekteyiz. İkinci bölümde, UKSB ağlarının kovaryans bilgisini kullanabilen türü olan Co-UKSB ağları ile verimli çevrimiçi öğrenme yöntemlerini incelemekteyiz. Kovaryans bilgisini UKSB ağının kapı yapısında kullanmakla birlikte çeşitli verimli UKSB modelleri sunmaktayız. Eklemeli gradyan güncellemeleri ve ağırlık matrisinin ayrıştırılması ile hesaplama karmaşıklığını düşürebilmekteyiz. Üçüncü bölümde, Co-UKSB ağlarının iletişim ağlarında saldırı tespiti için kullanımını göstermekteyiz. Dördüncü bölümde, Tree-UKSB ağlarının takviyeli sınıflandırılmış ve ikili hali olan BBT-UKSB ağlarını sunmaktayız. Tüm yapının dengeli bir biçimde geliştiği ve UKSB boğumlarının ikili bölündüğü N-li Tree-UKSB ağları için derinlik ve pencereleme etmenlerini takdim etmekteyiz. BBT-UKSB ağlarının hesaplama karmaşıklığını düşrmek için, ağırlık matrislerini ayrıştırmakta, çarpma işlemi yerine enerji verimli bir işlem kullanmakta ve BBT-UKSB ağının ağrılık matrislerinde tanımlı bir dilimleme işlemi tanımlamaktayız. Beşinci bölümde, girdi veri dizisi üzerinde en az sayıda atlamaya dayalı, düşük hesaplama karmaşıklığına sahip bir UKSB ağı sunmaktayız. Ardışık iki zaman arasındaki en uygun atlama mesafesini seçmek için iki adet yöntem üzerinde çalışmaktayız. Her bölümün sonunda, önerilen algoritmaların başarımını gerçek hayattan alınmış veri kümeleri ile yapılmış deneylerle göstermekteyiz.

Anahtar sözcükler: Çevrimiçi öğrenme, UKSB, kovaryans Tree-UKSB, takviyeli sınıflandırma, bağlanım, matris ayrıştırması.

Acknowledgement

I would first like to present my eternal gratitude to my advisor, Assoc. Prof. Suleyman Serdar Kozat, with my most sincere feelings. His priceless effort to guide me throughout my graduate studies helped me a lot to achieve this thesis. I am very grateful for completing my degree under his supervision. I believe that it would not be possible for me to achieve this work without his excellent support.

I would like to state my deep gratitude to Assoc. Prof. Tolga Cukur and Assist. Prof. Gokberk Cinbis for allocating their time to investigate my work and providing me with invaluable comments to make this thesis stronger.

Nobody has been more important to me in the pursuit of this achievement than the members of my family. I would like to thank my parents and my brothers and bhabhe's and brother's children whose love and guidance are with me in whatever I pursue. They are the ultimate role models. Love you all and I owe you my life.

I would also like to thank my Pakistani friends here at Bilkent for providing such a nice environment here no matter how far you are away form home.

Finally, I must express my very profound gratitude to Hira Noor(Hiru) for providing me with unfailing support and continuous encouragement throughout my years of MS study and through the process of researching and writing this thesis. No matter how much negative motivation and comments I got form the world, she was always a lighting beacon with endless positive support and motivation. This accomplishment would not have been possible without her. Thank you.

In the end, I would thank Allah The Almighty for bestowing His countless blessings on me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online learning is of vital importance and is widely studied in machine learning, adaptive signal processing and neural networks literature [1], [2], [3]. In most of the applications, nonlinear models are preferred over linear models. Neural networks based algorithms are highly used to model such nonlinear structures since they are capable to model highly nonlinear and complex structures. Although nonlinear approaches can be more effective and efficient than linear approaches in modelling, they usually suffer from overfitting problems, stability and convergence issues and have deteriorating performance [4], [5]. These problems are further exacerbated for big data applications where the input vectors are high dimensional. Nonlinear modelling of such high dimensional input vectors offers extreme challenges like high computational and time complexity with a deteriorating performance issue. Here, in particular, we study such nonlinear approaches in an online setting, where we receive input data sequences in a sequential manner along with its ground truth label. We aim to find a relation between them to predict future labels.

There exists a wide variety of nonlinear modelling approaches in the field of machine learning and big data signal processing literature [1], [6]. However, most of these approaches suffer from high time and computational complexity issues with

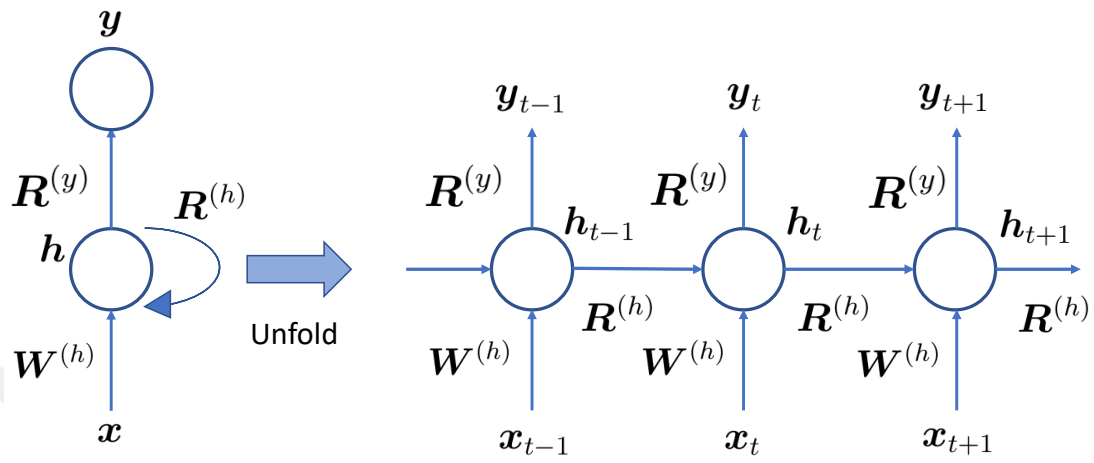

Figure 1.1: Schematic diagram of the RNN where the unfolded RNN is at the left side while the unfolded time version is shown on the right hand side of the figure.

a deteriorating performance. To remedy such issues, neural network based approaches are employed to model highly nonlinear structures. Such neural network algorithms are capable to model complex structure with enhanced performance [7], [8], [9],[10]. However, these algorithms are prone to various problems like overfitting, stability, and convergence [11], [12]. To mitigate these issues, deep neural networks (DNNs), i.e., a neural network with multiple layers, are introduced [13]. Introduction to DNNs significantly enhanced the overall performance while mitigating the issues previously encountered. However, DNNs are unable to handle the temporal data and fail to capture the time dependencies in the data. As a result, recurrent neural networks (RNNs) are introduced which not only handle the temporal data but also handle the time dependencies in the data very well [14], [15]. The schematic diagram of RNN is shown in Fig. 1.1.

The main problem with the RNNs is the exploding and vanishing gradients [11] as shown in Fig. 1.2. The LSTM architecture [12] solves the problem of learning long-term dependencies by introducing a memory cell that is able to preserve state over long periods of time. These LSTM networks have a gating structure that efficiently manages the long-term data dependency and provides an elegant solution to the exploding and vanishing gradient problem [12], [13]. The schematic diagram of the basic LSTM network is shown in Fig. 1.3.

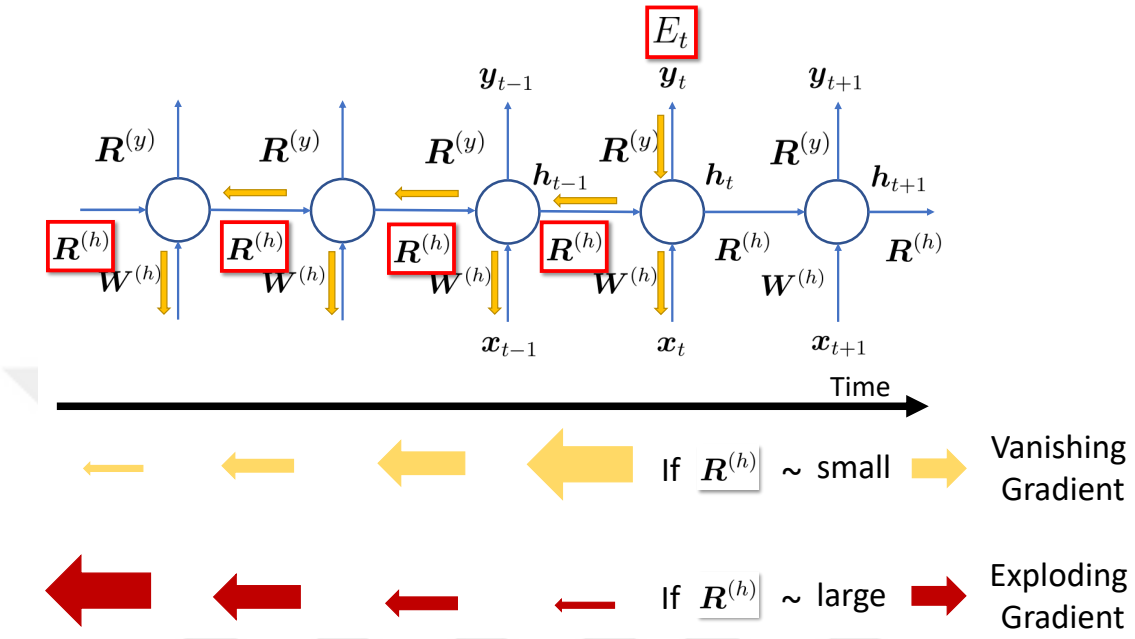

Figure 1.2: Vanishing and Exploding Gradient Problem in the RNN.

Training a neural network is one big task and is deeply investigated in machine learning literature. Most commonly, first-order gradient-based algorithms are used to train the corresponding parameters of the neural network. Various second order gradient-based algorithms are also used and give better performance than first-order methods but suffer from complexity issues [15]. Backpropagation Through Time (BPTT) is most commonly used and is highly efficient in computing gradients [15]. Additive updates are most commonly used in training the neural network parameters. Stochastic gradient descent (SGD) performs additive updates over the network parameters and is widely used. However, besides additive updates, multiplicative updates over the network parameters are used. Exponential gradient (EG) algorithm is used to perform multiplicative updates on the network parameters [16]. Performing either additive or multiplicative updates to train the complex nonlinear is an arduous task with high time and computational complexity.

In this thesis, we first introduce an efficient online learning algorithm using the LSTM networks. Since the gating mechanism of the LSTM networks is its backbone, we incorporate additional information into the gating mechanism that

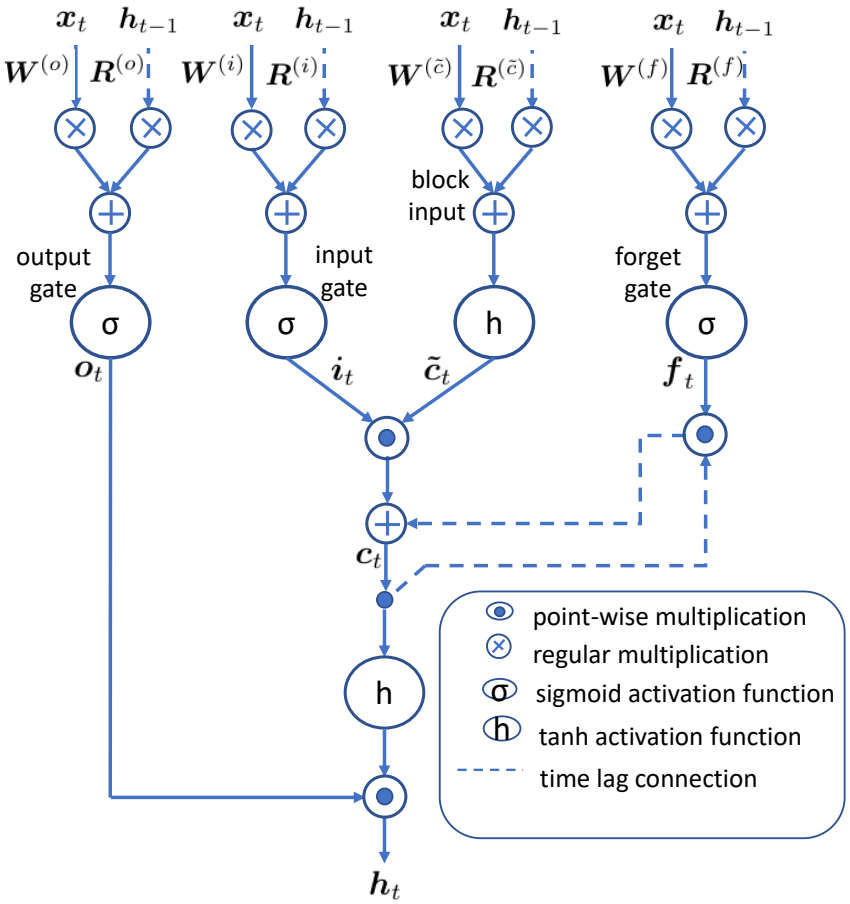

Figure 1.3: Schematic diagram of the LSTM Network.

helps in learning and modelling complex processes efficiently [10], [11], [12]. We consider the covariance of the present and past input to the LSTM networks and add their effect to the gating mechanism of the LSTM networks. Besides adding the input covariance, we also consider the output covariance information and add it to the gating mechanism of the LSTM networks. We then add adjustable weights to these input and output covariance information. These weights help in learning efficiently during the training process and helps in increasing the overall performance. We call this modified covariance information based LSTM networks as Co-LSTM networks.

Although the addition of covariance information into the gating structure of the LSTM networks makes it more intelligent this comes at the cost of an increase in time and computational complexity. This increase in time and computational

complexity results from the additional input and output weight and covariance matrices of the Co-LSTM networks. In order to make the resulting network to have low complexity, we use the weight matrix factorization (WMF) trick on the Co-LSTM network weight matrices [17]. The objective of the WMF is to break down a higher rank matrix into two lower rank matrices. The use of WMF on the Co-LSTM network weight matrices significantly reduces the time and computational complexity besides maintaining the overall accuracy or performance of the system.

There are many applications like genetic protein therapy, housing prices etc [18] where there is some direct or indirect link between the past and present values. Knowing this past and present relation can help in future predictions. This past and the present relation are grasped using the covariance. Adding this covariance information to the system helps in more detailed learning of the process no matter how complex the process is. We utilize this covariance information in our LSTM networks in their gating structures. Since the gating mechanism controls the flow of information, we make use of this gating mechanism and add or modify the input and output covariance information in the LSTM networks. Intuitively, the input and output covariance information in the gating architecture efficiently controls the flow of the information that results in increased performance. The percentage of addition or subtraction of the covariance information is being learned during the training process via their covariance weight matrices.

We then develop and implement regression-based LSTM networks using a tree-based architecture. The main purpose of using the tree-based structure is to add the boosting effect in the learning process. In [19], Tree-LSTM networks is introduced that efficiently captures the structural information. This structural information is of utmost importance in the sentiment analysis of the tweets and in analyzing the stock price data. The tree-based LSTM trees have an upper hand over the simple LSTM networks in terms of multiple dependencies. This means that the LSTM gating vectors and memory cell updates depend on multiple LSTM units which are called as child LSTM nodes or units. Moreover, each parent LSTM node incorporates the combined effect of all the forget gates of its child LSTM nodes. There are two tree-based LSTM structures proposed in

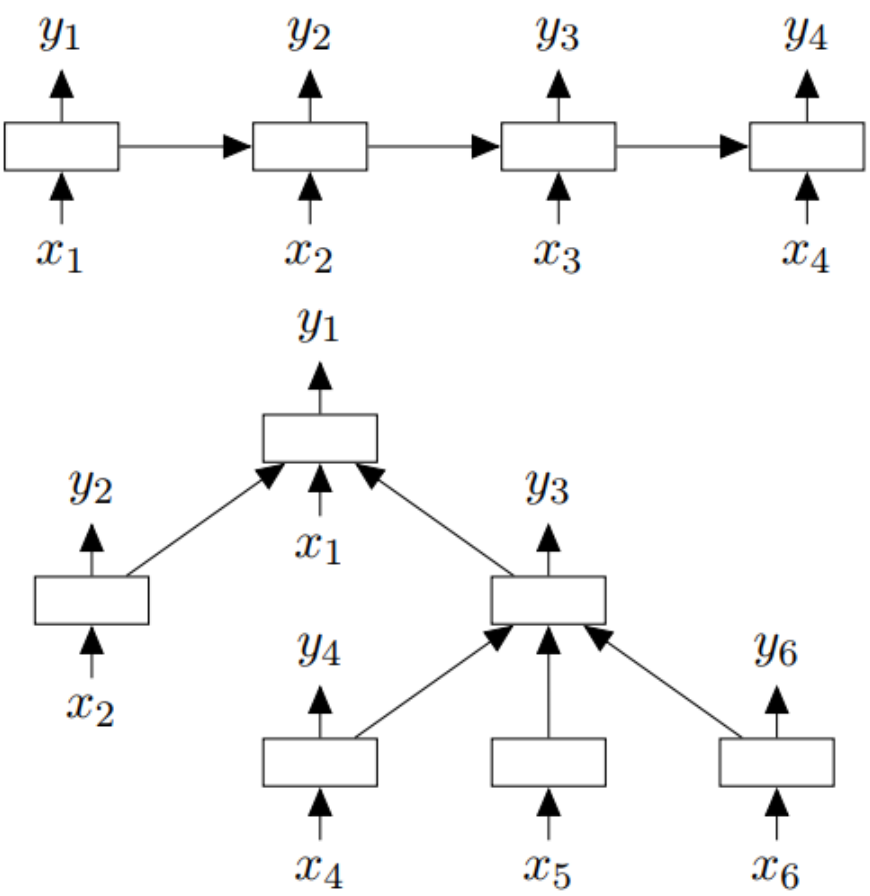

Figure 1.4: Schematic diagram of the Tree-LSTM networks where at Top is the regular Chain structured LSTM networks and at Bottom is the basic framework for the Tree-LSTM networks.

[19], [20] namely; Child-Sum Tree-LSTMs and N-ary Tree-LSTMs. The basic Tree-LSTM framework is shown in Fig. 1.4.

N-ary Tree LSTMs are introduced in [19], [20] that skips the sub-trees using their forget gates response that has a very little effect. Tree-LSTM networks updates or manages its hidden state value based on the input and forget gate response from its corresponding child [19], [20]. Whereas, in the simple LSTM networks, the hidden state at a current time is handled by the hidden state from the previous time step. Simple LSTM networks can be derived from the Tree-LSTM networks, i.e., they are a subset of Tree-LSTM networks.

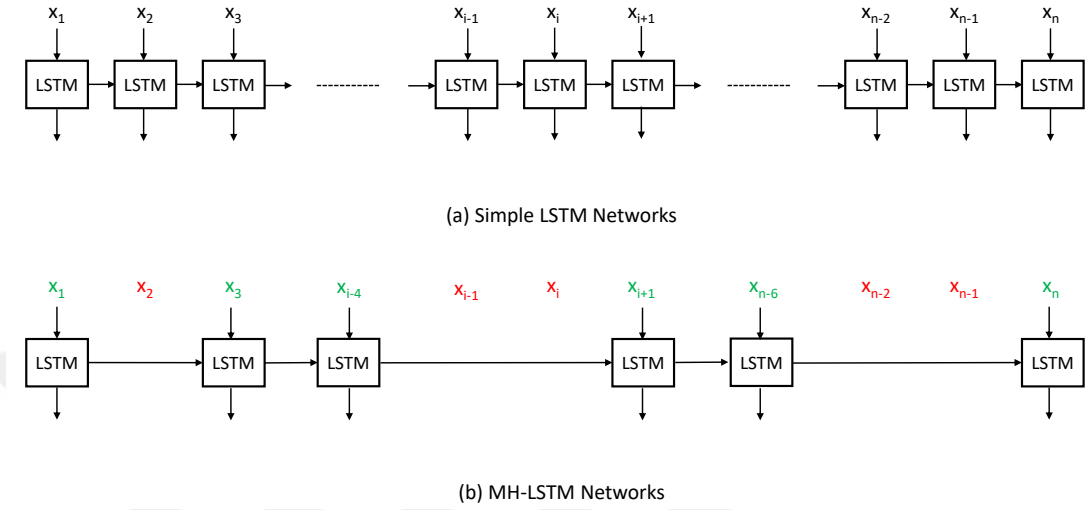

Figure 1.5: (a) Simple LSTM networks applied over the entire input sequence $\{\boldsymbol{x}_t\}_{t=1}^{n}$ and (b) MH-LSTM networks over the input sequence $\{\boldsymbol{x}_t\}_{t=1}^{n}$ with variable hopping.

In this thesis, we specifically modify the variant of the Tree-LSTM networks, i.e., N-ary Tree-LSTM networks. We introduce a boosted version of the Tree-LSTM networks. Instead of splitting the parent LSTM node into N child nodes, we split the parent LSTM node into two child LSTM nodes, i.e., binary splitting. We let the tree grow by binary splitting each node in a balanced manner thus obtaining a balanced tree in the end. The depth of the tree is denoted by α. We also introduce an input windowing factor that takes a fraction of input and insert into the child LSTM nodes according to each distinct depth of the tree. The information flow is from bottom to top and is in a hierarchical manner. While moving from bottom to top, we supply the previous depth child node forget gate response to the upper parent LSTM node. Each parent LSTM node takes in total two bottom child LSTM node forget gate response and process in such a way that it ignores the child LSTM node with a little effect. This process continues until we reach the ultimate parent LSTM node. The final decision is a combined effect that constitutes the forget gate response of all the child LSTM nodes. This depth factor and inclusion of input windowing factor to the child nodes are responsible for boosting the result of the network. We validate the performance of the proposed algorithms on real-life data sets. We use mean square error (MSE) as performance criteria. We also take into account the total

training time it requires to train the corresponding parameters.

Finally, we introduce another low complexity variant of the LSTM networks. Since we know that the LSTM networks unroll in time over the whole input data sequence; the amount of time it takes for training a model for very long input data sequences can be very high. This results in an increase in time and computational complexity. In order to significantly reduce the complexity and still achieving the best performance, we introduce a low complexity version of the LSTM networks that operates by employing a minimum number of hops over the input data sequence. We call these low complexity networks as Minimal Hop Long Short Term Memory (MH-LSTM) networks as shown in Fig. 1.5. These MH-LSTM networks have low time and computational complexity. The main idea is to reduce the total number of hops over the input data sequence significantly but still achieving nearest to the best possible performance. For example, let us say we have an input data sequence of length 100, then instead of hopping over each time instance, i.e., 100 times, we hop let us say 10 time which is ten times less than the original one and still achieving closest to the best overall performance.

Furthermore, we consider the regression performances of the algorithms based on the Gated Recurrent Unit (GRU) networks. In the previous sections, we use the LSTM architecture. Since our approach is generic, we also apply our approach to the recently introduced GRU architecture. We list down the results using the GRU networks in Appendix B, C and D.

1.1 Thesis Contribution

The main contributions of this thesis are as follows:

- We propose an efficient online learning algorithm using the LSTM networks by introducing the covariance information of the input data sequence into the gating structure of the LSTM networks. We call these networks as Co-LSTM networks.

- Besides using the covariance information for the input vector in the gating structure of the LSTM networks, we also include the covariance of the output vector of the LSTM networks. We then also add adjustable weight to the covariance matrix to help in the learning process.

- In order to alleviate the time and computational complexity, we applied the weight matrix factorization trick to the network weight matrices of the Co-LSTM networks. We then illustrate the performance of the Co-LSTM networks using real-life data sets and show that the Co-LSTM networks perform better than the Simple-LSTM networks.

- We then propose a boosted binary version of Tree LSTM regression networks which we call BBT-LSTM networks. We introduce depth and input windowing factor in the simple N-ary Tree LSTM networks and restrict the splittings at each parent/child node to two and let the tree grow in a balanced manner.

- We introduce slicing operation into the BBT-LSTM networks that significantly reduces the time and computational complexity. Besides slicing operation, we further reduce the time and computational complexity by employing energy efficient multiplication operator and weight matrix factorization on the BBT-LSTM network matrices.

- The boosting effect in the BBT-LSTM networks is achieved from bottom to top via each parent LSTM node taking in total two bottom child LSTM node forget gate response and process in such a way that it ignores the child LSTM node with a little effect. This process continues until we reach the ultimate parent LSTM node.

- Through an extensive set of experiments on real-life data sets, we demonstrate that the proposed BBT-LSTM networks perform significantly well as compared to simple LSTM and tree LSTM networks.

- We then implement a low complexity version of the LSTM networks that operates by employing a minimum number of hops over the input data sequence. We call these low complexity networks as MH-LSTM networks.

These MH-LSTM networks have low time and computational complexity and still achieving the closest possible best overall performance.

1.2 Thesis Outline

This thesis is divided into six chapters as follows. In Chapter 2, we introduce input and output covariance information into the gating structure of the LSTM networks and call them Co-LSTM networks. We define the covariance matrix and introduce its effect into the LSTM networks. We then derive the additive gradient based updates for the Co-LSTM networks. In order to reduce the complexity of the Co-LSTM network, we apply the WMF trick on the weight matrices of the Co-LSTM networks. We then again derive the additive gradient based updates for the WMF based Co-LSTM networks. We then illustrate the performance of the Co-LSTM networks using the real-life data sets.

In Chapter 3, we use more practical and real data set in which we execute network intrusion detection using the proposed Co-LSTM networks in Chapter 2. We give both supervised and unsupervised frameworks for network intrusion detection. For the supervised framework, we use two configurations of the Co-LSTM networks (one with separate effects of source and destination payloads and second with the ongoing continuous effect of source and destination payloads) with three types of pooling layer namely; mean, max and last pooling. For the unsupervised framework, we develop a sequential Co-LSTM encoder-decoder frameworks that detect the anomalies in the network payloads by keeping an eye on the reconstruction decoder error function with a threshold. We further illustrate the performance of the proposed algorithm with various other simple and conventional algorithms.

In Chapter 4, we then introduce a tree-based structure of the LSTM networks. We implement boosted binary regression Tree-LSTM networks with low complexity. We propose several methods to reduce the total time and computational complexity of the BBT-LSTM networks like; WMF based BBT-LSTM networks,

ef-operator based BBT-LSTM networks and ef-WMF based BBT-LSTM networks. Furthermore, we introduce a slicing operation over the network weight matrices to reduce the complexity of the BBT-LSTM networks. We then demonstrate the performance of the BBT-LSTM networks using various real-life data sets.

In Chapter 5, we implement another low complexity based version of the LSTM networks based on hopping over the input data sequence. We propose two methods for efficient selection of hopping distance and give its detail which is then followed by computational and time complexity and performance trade-offs. Finally, we illustrate the performance of the MH-LSTM networks by using real-life data sets.

Finally, in Chapter 6, we give the concluding remarks.

Chapter 2

Covariance Information Based Efficient Online Learning Using LSTM Neural Networks

In this part of the thesis, we develop and implement efficient online learning algorithms using the LSTM neural networks. We introduce highly efficient and intelligent online learning algorithms using the LSTM neural networks that employ additional yet useful information. This useful information is the covariance of the present and one-time step past input vector that is added and modified to the gating structure of the LSTM networks. Besides using the covariance information for the input vector in the gating structure of the LSTM networks, we also include the covariance of the output (state) vector of the LSTM networks. We call these covariance based LSTM networks as Co-LSTM networks. We then also add adjustable weight to the covariance matrix to help in the learning process. In order to alleviate the time and computational complexity, we applied the weight matrix factorization (WMF) trick to the network weight matrices [17]. In the end, we illustrate the performance of the Co-LSTM networks using real-life data sets and show that the Co-LSTM networks perform better than the Simple-LSTM networks.

2.1 Problem Statement

We sequentially receive input vector $\{\boldsymbol{x}_t\}_{t\geq 1}$, $\boldsymbol{x}_t \in \mathbb{R}^p$ along with desired output $\{d_t\}_{t\geq 1}$. At time t, our aim is to find $\hat{d}_t$ and then evaluate the loss function as

$$\ell_t(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2.$$

In order to calculate the value of $\hat{d}_t$, we use the basic architecture of the RNNs [14], [15] defined as follows:

$$\begin{aligned}\boldsymbol{h}_t &= \gamma(\boldsymbol{W}^{(h)}\boldsymbol{x}_t + \boldsymbol{R}^{(h)}\boldsymbol{h}_{t-1})\\ \boldsymbol{y}_t &= \beta(\boldsymbol{R}^{(y)}\boldsymbol{h}_t),\end{aligned}$$

where $\boldsymbol{h}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output vector of the RNN. The functions $\gamma(\cdot)$ and $\beta(\cdot)$ apply pointwise to the vectors and are commonly set to $\tanh(\cdot)$. The coefficient weight matrices have the dimensions $\boldsymbol{W}^{(h)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(h)} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{R}^{(y)} \in \mathbb{R}^m$.

We specifically use the LSTM networks as a special case of the RNNs in order to calculate the estimate of the desired output, i.e., $\hat{d}_t$. The LSTM network with one hidden layer is defined as follows [12]:

$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{2.1}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\right) \tag{2.2}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}\right) \tag{2.3}$$

$$\boldsymbol{c}_t = \boldsymbol{\Pi}_t^{(\boldsymbol{i})}\tilde{\boldsymbol{c}}_t + \boldsymbol{\Pi}_t^{(\boldsymbol{f})}\boldsymbol{c}_{t-1} \tag{2.4}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}\right) \tag{2.5}$$

$$\boldsymbol{h}_t = \boldsymbol{\Pi}_t^{(\boldsymbol{o})}\tanh(\boldsymbol{c}_t), \tag{2.6}$$

where $\boldsymbol{c}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector and $\boldsymbol{h}_t \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$ and $\boldsymbol{o}_t$ are the input, forget and output gates, respectively. Also, $\boldsymbol{\Pi}_t^{(\boldsymbol{i})} = diag(\boldsymbol{i}_t)$, $\boldsymbol{\Pi}_t^{(\boldsymbol{f})} = diag(\boldsymbol{f}_t)$ and $\boldsymbol{\Pi}_t^{(\boldsymbol{o})} = diag(\boldsymbol{o}_t)$ where $diag(\cdot)$ is the diagonal matrix. The sigmoid function $\sigma(\cdot)$ and $\tanh(\cdot)$ applies pointwise to the vector elements. The weight matrices and bias vectors have

the following dimensions, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m \times p}$, $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{b}^{(\cdot)} \in \mathbb{R}^{m}$. We calculate the final estimate of the desired output as

$$\hat{d}_t = \boldsymbol{p}_t^T \boldsymbol{h}_t \tag{2.7}$$

where $\boldsymbol{p}_t \in \mathbb{R}^m$.

2.1.1 Covariance Matrix based LSTM Networks (Co-LSTM)

In this subsection, we first define the covariance matrix of $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$ as $\sum_x = Cov(\boldsymbol{x}_t, \boldsymbol{x}_{t-1})$ where $\sum_x \in \mathbb{R}^{(p \times p)}$. We will now add this covariance information to the gating part of the original LSTM networks. We propose three different models for the covariance information based LSTM networks. Note that we only show the changes made to the LSTM network for the input gate, while rest of the changes for the other gate equations are the same. The basic schematic diagram of the Co-LSTM networks is shown in Fig. 2.1.

2.1.1.1 Model 1 : Additional Weighted Input and Output Covariance Information Model

In our first model, we add the weighted input and output covariance information to the gating structure of the simple LSTM network. For instance, the input gate is represented as

$$\boldsymbol{i}_t = \sigma\Big(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)} + \boldsymbol{T}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}^{(i)} \odot \sum_h \boldsymbol{h}_{t-1}\Big), \tag{2.8}$$

where $\boldsymbol{T}^{(i)} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{S}^{(i)} \in \mathbb{R}^{m \times m}$ are the weight matrices for the covariance matrices for input and output vectors respectively. In (2.8), $\sum_x \in \mathbb{R}^{m \times p}$ and $\sum_h \in \mathbb{R}^{m \times m}$ are the input and output covariance matrices respectively. Similarly, we can write the remaining forget and output gate LSTM equations following the above structure.

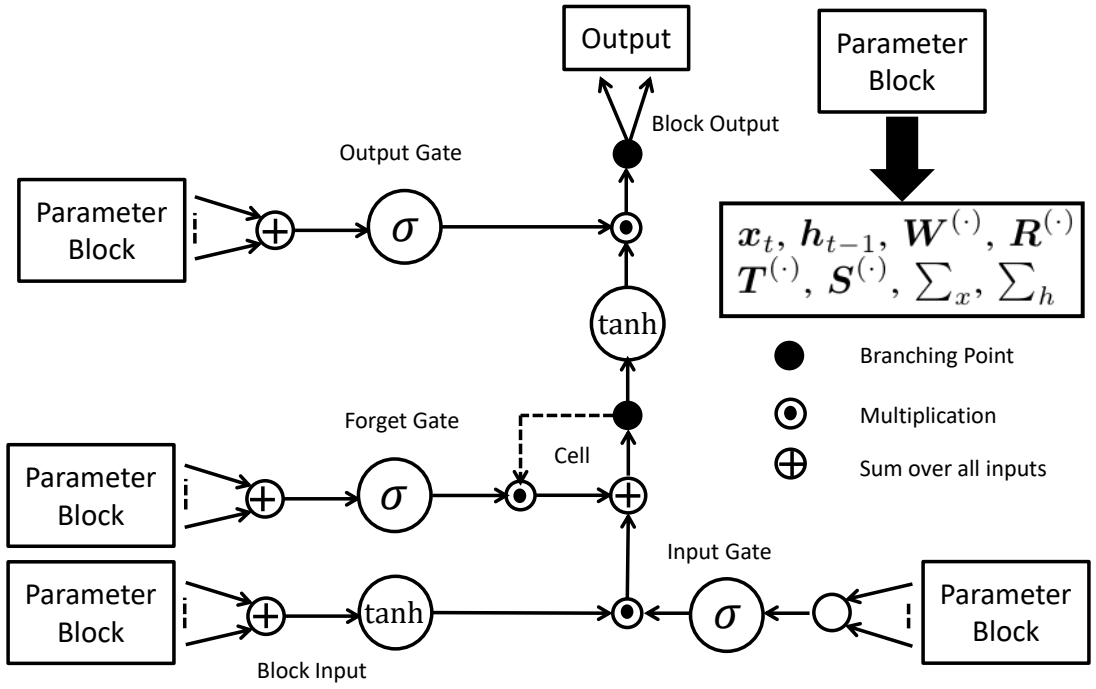

Figure 2.1: Schematic diagram of the Co-LSTM networks. Note that instead of $\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$ and simple-LSTM network parameters like $\boldsymbol{W}^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)}$, we have additional covariance information parameters like $\boldsymbol{T}^{(\cdot)}$, $\boldsymbol{S}^{(\cdot)}$, $\sum_x$ and $\sum_h$.

2.1.1.2 Model 2 : Weighted Input Covariance Model

In our second model, we modify the simple weighted input with the weighted covariance information to the gating part of the simple LSTM network. The input gate is written as

$$\boldsymbol{i}_t = \sigma\Big(\boldsymbol{W}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\Big). \tag{2.9}$$

2.1.1.3 Model 3 : Weighted Input and Output Covariance Model

In our third model, we modify the simple weighted input with the weighted covariance information to the simple LSTM network. The input gate is written as

$$\boldsymbol{i}_t = \sigma\Big(\boldsymbol{W}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{R}^{(i)} \odot \sum_h \boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\Big). \tag{2.10}$$

Remark 2.1: *In the simple LSTM networks, at time t, we have both the past and present information available. In our proposed algorithm, we add the covariance information of the past and present input. This is helpful because in some applications and data sets like protein therapy, stock exchange etc the past and present input are interlinked and their covariance information helps in adding more information to the network and hence resulting in an increase in output performance.*

2.1.2 Additive Gradient Based Update Calculations for the Co-LSTM Networks

In this subsection, we perform additive based gradient updates using the stochastic gradient descent (SGD) algorithm [21]. As defined before, for the loss function $\ell_t(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$, the SGD update for the parameter α is calculated as

$$\alpha_{t+1} = \alpha_t + \mu \frac{\partial \ell_t}{\partial \alpha} = \alpha_t - 2\mu(d_t - \hat{d}_t)\boldsymbol{p}^T \frac{\partial \boldsymbol{h}_t}{\partial \alpha}, \tag{2.11}$$

where μ is the learning rate parameter and $\dfrac{\partial \boldsymbol{h}_t}{\partial \alpha}$ is calculated as follows

$$\frac{\partial \boldsymbol{h}_t}{\partial \alpha} = \boldsymbol{\Pi}_t^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_t) + \boldsymbol{\Pi}_t^{(\boldsymbol{o})} \boldsymbol{\Pi}_t^{(\tanh'(\boldsymbol{c}_t))} \frac{\partial \boldsymbol{c}_t}{\partial \alpha}, \tag{2.12}$$

where $\dfrac{\partial \boldsymbol{c}_t}{\partial \alpha}$ and $\boldsymbol{o}' = \dfrac{\partial \boldsymbol{o}_t}{\partial \alpha}$ is written as follows

$$\frac{\partial \boldsymbol{c}_t}{\partial \alpha} = \boldsymbol{\Pi}_t^{(\boldsymbol{i}')} \tilde{\boldsymbol{c}}_t + \boldsymbol{\Pi}_t^{(\boldsymbol{i})} \boldsymbol{\Pi}_t^{(\tanh'(\boldsymbol{\beta}))} \frac{\partial \boldsymbol{\beta}}{\partial \alpha} + \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{f}')} \boldsymbol{c}_{t-1} + \boldsymbol{\Pi}_t^{(\boldsymbol{f})} \boldsymbol{\gamma}_{c_{t-1}}^{(\alpha)} \tag{2.13}$$

$$\frac{\partial \boldsymbol{o}_t}{\partial \alpha} = \boldsymbol{\Pi}_t^{(\sigma'(\boldsymbol{\rho}))} \frac{\partial \boldsymbol{\rho}}{\partial \alpha} \tag{2.14}$$

where

$$\boldsymbol{\beta} = \boldsymbol{W}^{(i)} \boldsymbol{x}_t + \boldsymbol{R}^{(i)} \boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)} + \boldsymbol{T}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}^{(i)} \odot \sum_h \boldsymbol{h}_{t-1}$$

$$\boldsymbol{\rho} = \boldsymbol{W}^{(o)} \boldsymbol{x}_t + \boldsymbol{R}^{(o)} \boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)} + \boldsymbol{T}^{(o)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}^{(o)} \odot \sum_h \boldsymbol{h}_{t-1}$$

and

$$\boldsymbol{\gamma}_{c_{t-1}}^{(\alpha)} = \frac{\partial \boldsymbol{c}_{t-1}}{\partial \alpha}.$$

Specifically, for $\alpha = w_{ij}^{(i)}$, the gradient based updates as stated in (2.12), (2.13) and (2.14) can be written as follows

$$\frac{\partial \boldsymbol{h}_t}{\partial w_{ij}^{(i)}} = \boldsymbol{\Pi}_t^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_t) + \boldsymbol{\Pi}_t^{(\boldsymbol{o})} \boldsymbol{\Pi}_t^{(\tanh'(\boldsymbol{c}_t))} \frac{\partial \boldsymbol{c}_t}{\partial w_{ij}^{(i)}} \tag{2.15}$$

where $\boldsymbol{\Pi}_t^{(\boldsymbol{o}')} = diag\left(\frac{\partial \boldsymbol{o}_t}{\partial w_{ij}^{(i)}}\right)$ and $\frac{\partial \boldsymbol{o}_t}{\partial w_{ij}^{(i)}}$, $\frac{\partial \boldsymbol{c}_t}{\partial w_{ij}^{(i)}}$ can be explicitly written as

$$\begin{aligned}\frac{\partial \boldsymbol{o}_t}{\partial w_{ij}^{(i)}} = \boldsymbol{\Pi}_t^{(\sigma'(\boldsymbol{\rho}))} \Bigg(& \boldsymbol{R}^{(o)} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{R}^{(o)} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \\ & \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \boldsymbol{\gamma}_{ij,t-1}^{(i)} + \boldsymbol{A}'^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \tanh(\boldsymbol{c}_{t-1}) + \\ & \boldsymbol{A}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{A}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \\ & \boldsymbol{\gamma}_{ij,t-1}^{(i)} \Bigg)\end{aligned} \tag{2.16}$$

and

$$\frac{\partial \boldsymbol{c}_t}{\partial w_{ij}^{(i)}} = \boldsymbol{\Pi}_t^{(\tilde{\boldsymbol{c}})} \frac{\partial \boldsymbol{i}_t}{\partial w_{ij}^{(i)}} + \boldsymbol{\Pi}_t^{(\boldsymbol{i})} \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial w_{ij}^{(i)}} + \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{c})} \frac{\partial \boldsymbol{f}_t}{\partial w_{ij}^{(i)}} + \boldsymbol{\Pi}_t^{(\boldsymbol{f})} \boldsymbol{\gamma}_{ij,t-1}^{(i)}, \tag{2.17}$$

where $\boldsymbol{A}^{(\cdot)} = \boldsymbol{S}^{(\cdot)} \odot \sum_h$ and

$$\begin{aligned}\frac{\partial \boldsymbol{i}_t}{\partial w_{ij}^{(i)}} = \boldsymbol{\Pi}_t^{(\boldsymbol{\theta})} \Bigg(& \boldsymbol{\Delta}_{ij} \boldsymbol{x}_t + \boldsymbol{R}^{(i)} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{R}^{(i)} \\ & \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \boldsymbol{\gamma}_{ij,t-1}^{(i)} + \boldsymbol{A}'^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \tanh(\boldsymbol{c}_{t-1}) + \\ & \boldsymbol{A}^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{A}^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \\ & \boldsymbol{\gamma}_{ij,t-1}^{(i)} \Bigg),\end{aligned} \tag{2.18}$$

where $\boldsymbol{\Delta}_{ij}$ is a matrix of zeros except 1 at the ith row and jth column, while rest of the gradients are similar to (2.18) with $\boldsymbol{\Delta}_{ij} = \mathbf{0}$ and $\boldsymbol{\theta} = \boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)} + \boldsymbol{T}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}^{(i)} \odot \sum_h \boldsymbol{h}_{t-1}$.

2.1.3 Weight Matrix Factorization based Co-LSTM Networks (WMF-Co-LSTM)

In this subsection, we introduce the WMF based Co-LSTM networks (WMF-Co-LSTM). Since the introduction of covariance information in the gating structure of the LSTM networks increases the total number of trainable parameters, we employ the concept used in [17], where we decompose a high rank matrix into two smaller rank matrices. Let us say, for a matrix $\boldsymbol{D} \in \mathbb{R}^{m\times p}$, where m and p are large numbers, i.e., a higher rank matrix, we can write $\boldsymbol{D} \approx \boldsymbol{D}_1\boldsymbol{D}_2$ where $\boldsymbol{D}_1 \in \mathbb{R}^{m\times q}$ and $\boldsymbol{D}_2 \in \mathbb{R}^{q\times p}$ such that

$$q << \min(m, p).$$

We use the WMF approach on the weight matrices of all the models of the Co-LSTM networks, i.e., $\boldsymbol{W}^{(\cdot)} \approx \boldsymbol{W}_1^{(\cdot)}\boldsymbol{W}_2^{(\cdot)}$, $\boldsymbol{R}^{(\cdot)} \approx \boldsymbol{R}_1^{(\cdot)}\boldsymbol{R}_2^{(\cdot)}$, $\boldsymbol{S}^{(\cdot)} \approx \boldsymbol{S}_1^{(\cdot)}\boldsymbol{S}_2^{(\cdot)}$ and $\boldsymbol{T}^{(\cdot)} \approx \boldsymbol{T}_1^{(\cdot)}\boldsymbol{T}_2^{(\cdot)}$.

Proposition 2.1: *The total number of parameters of Co-LSTM networks with the WMF are less than the total number of parameter of the Co-LSTM networks without the WMF.*

Proof: Let $P_{(WMF-Co-LSTM)}$ and $P_{(Co-LSTM)}$ be the total number of parameters of the Co-LSTM networks with and without the WMF. Note that, here we only show the calculations for the Model 1 of the Co-LSTM networks. The

calculations of the other two models follow the same pattern. We have

$$\begin{aligned}
\frac{P_{(WMF-Co-LSTM)}}{P_{(Co-LSTM)}} &= \frac{4\big(2(pq+qm+2mn)+m\big)}{4\big(2(pm+mm)+m\big)} \\
&= \frac{2q(p+m)+4mn+m}{2m(p+m)+m} \\
&\approx \frac{2q(p+m)+4mn}{2m(p+m)} \\
&\approx \frac{2q(p+m)}{2m(p+m)} + \frac{4mn}{2m(p+m)} \\
&\approx \frac{q}{m} + \frac{2n}{p+m} \qquad (2.19)
\end{aligned}$$

Since $q << \min(m,p)$ and $n << m$, (2.19) implies that $P_{(WMF-Co-LSTM)} < P_{(Co-LSTM)}$.

□

Table 2.1: Total number of parameters of the Simple-LSTM, Co-LSTM (Model 1) and WMF-Co-LSTM (Model 1) Networks. Here, $q << \min(m,p)$ and $n << m$.

Algorithms	**Number of Parameters**
Simple-LSTM	$4\big(pm+mm+m\big)$
Co-LSTM (Model 1)	$4\big(2(pm+mm)+m\big)$
WMF-Co-LSTM (Model 1)	$4\big(2(pq+qm+2mn)+m\big)$

In Table 2.1, we list down the total number of parameters of the Simple-LSTM networks without WMF and Co-LSTM networks (Model 1) with and without WMF. We observe in Table 2.1 that the total number of parameters for the Co-LSTM networks (specifically Model 1) increases by two-fold as compared to the total number of parameters for the Simple-LSTM networks. So in order to alleviate the total number of parameters for the Co-LSTM networks, we used the WMF approach that drastically reduces the total number of parameters while still maintaining the accuracy.

Remark 2.2: *The total number of parameters for the Model 2 and Model 3 for the Co-LSTM networks are the same as that of Simple-LSTM networks. Model 2 and Model 3 possess an upper hand in terms of additional covariance information while having the same number of parameters as in Simple-LSTM networks. We analyze and compare the performance of the Co-LSTM networks with the Simple-LSTM networks in Sec. 2.2.*

2.1.4 Additive Gradient Based Update Calculations for WMF-Co-LSTM Networks

In this subsection, we perform gradient based calculations for the Co-LSTM networks (specifically Model 1) with the WMF. For instance, the input gate of the Model 1 of the Co-LSTM networks can be written as

$$\begin{aligned}\boldsymbol{i}_t = \sigma\Big(&\boldsymbol{W}_1^{(i)}\boldsymbol{W}_2^{(i)}\boldsymbol{x}_t + \boldsymbol{R}_1^{(i)}\boldsymbol{R}_2^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)} + \\ &\boldsymbol{T}_1^{(i)}\boldsymbol{T}_2^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}_1^{(i)}\boldsymbol{S}_2^{(i)} \odot \sum_h \boldsymbol{h}_{t-1}\Big).\end{aligned} \tag{2.20}$$

Rest of the gating equations can be written similarly following the above pattern. Following the procedure as in 2.1.2, for $\alpha = w_{1_{ij}}^{(i)}$, where $w_{1_{ij}}^{(i)}$ is the ith row and jth column entry of $\boldsymbol{W}_1^{(i)}$, the gradient based updates as stated in (2.12), (2.13) and (2.14) can be written as follows

$$\frac{\partial \boldsymbol{h}_t}{\partial w_{1_{ij}}^{(i)}} = \boldsymbol{\Pi}_t^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_t) + \boldsymbol{\Pi}_t^{(\boldsymbol{o})} \boldsymbol{\Pi}_t^{(\tanh'(\boldsymbol{c}_t))} \frac{\partial \boldsymbol{c}_t}{\partial w_{1_{ij}}^{(i)}} \tag{2.21}$$

where $\boldsymbol{\Pi}_t^{(\boldsymbol{o}')} = diag\left(\dfrac{\partial \boldsymbol{o}_t}{\partial w_{1_{ij}}^{(i)}}\right)$ and $\dfrac{\partial \boldsymbol{o}_t}{\partial w_{1_{ij}}^{(i)}}$, $\dfrac{\partial \boldsymbol{c}_t}{\partial w_{1_{ij}}^{(i)}}$ can be explicitly written as

$$\begin{aligned}\frac{\partial \boldsymbol{o}_t}{\partial w_{1_{ij}}^{(i)}} = \boldsymbol{\Pi}_t^{(\sigma'(\boldsymbol{\rho}))}\Bigg(&\boldsymbol{R}_1^{(o)}\boldsymbol{R}_2^{(o)}\boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{R}_1^{(o)}\boldsymbol{R}_2^{(o)} \\ &\boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})}\boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))}\boldsymbol{\gamma}_{ij,t-1}^{(i)} + \boldsymbol{\Theta}'^{(\boldsymbol{o})}\boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \tanh(\boldsymbol{c}_{t-1}) + \\ &\boldsymbol{\Theta}^{(\boldsymbol{o})}\boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{\Theta}^{(\boldsymbol{o})}\boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})}\boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \\ &\boldsymbol{\gamma}_{ij,t-1}^{(i)}\Bigg)\end{aligned} \tag{2.22}$$

and

$$\frac{\partial \boldsymbol{c}_t}{\partial w_{1_{ij}}^{(i)}} = \boldsymbol{\Pi}_t^{(\tilde{\boldsymbol{c}})} \frac{\partial \boldsymbol{i}_t}{\partial w_{1_{ij}}^{(i)}} + \boldsymbol{\Pi}_t^{(\boldsymbol{i})} \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial w_{1_{ij}}^{(i)}} + \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{c})} \frac{\partial \boldsymbol{f}_t}{\partial w_{1_{ij}}^{(i)}} + \boldsymbol{\Pi}_t^{(\boldsymbol{f})} \boldsymbol{\gamma}_{ij,t-1}^{(i)},$$

where $\boldsymbol{\Theta}^{(\cdot)} = \boldsymbol{S}_1^{(\cdot)} \boldsymbol{S}_2^{(\cdot)} \odot \sum_h$ and

$$\begin{aligned} \frac{\partial \boldsymbol{i}_t}{\partial w_{1_{ij}}^{(i)}} = \boldsymbol{\Pi}_t^{(\boldsymbol{\theta})} \Bigg(& \boldsymbol{\Delta}_{ij} \boldsymbol{x}_t + \boldsymbol{R}_1^{(i)} \boldsymbol{R}_2^{(i)} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{R}_1^{(i)} \\ & \boldsymbol{R}_2^{(i)} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \boldsymbol{\gamma}_{ij,t-1}^{(i)} + \boldsymbol{\Theta}'^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \tanh(\boldsymbol{c}_{t-1}) \\ & + \boldsymbol{\Theta}^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o}')} \tanh(\boldsymbol{c}_{t-1}) + \boldsymbol{\Theta}^{(\boldsymbol{i})} \boldsymbol{\Pi}_{t-1}^{(\boldsymbol{o})} \boldsymbol{\Pi}_{t-1}^{(\tanh'(\boldsymbol{c}))} \\ & \boldsymbol{\gamma}_{ij,t-1}^{(i)} \Bigg), \end{aligned} \tag{2.23}$$

where $\boldsymbol{\Delta}_{ij}$ is a matrix of zeros except 1 at the ith row and jth column, while rest of the gradients are similar to (2.23) with $\boldsymbol{\Delta}_{ij} = \boldsymbol{0}$ and $\boldsymbol{\theta} = \boldsymbol{W}^{(i)} \boldsymbol{x}_t + \boldsymbol{R}^{(i)} \boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)} + \boldsymbol{T}^{(i)} \odot \sum_x \boldsymbol{x}_t + \boldsymbol{S}^{(i)} \odot \sum_h \boldsymbol{h}_{t-1}$.

2.2 Experiments

2.2.1 Regression Data Sets

In this section, we illustrate the performance of the proposed models using real-life data sets, i.e., Kinematics [22], Elevators [23], Protein Tertiary [24] and Puma8NH [25] data sets. We compare the performance of all the three proposed Co-LSTM models with simple LSTM networks. For a fair experimental setup, we keep the parameters same in all the set of experiments for all the models.

In Fig. 2.2, we compare the regression performance for the Kinematics data set [22] using all the models of the proposed Co-LSTM networks and Simple-LSTM networks. In this data set, we have a five-dimensional input vector for the robotic arm and we need to predict the distance based on the feedback to the robotic arm. Upon experiments, we observe that as we add the additional covariance

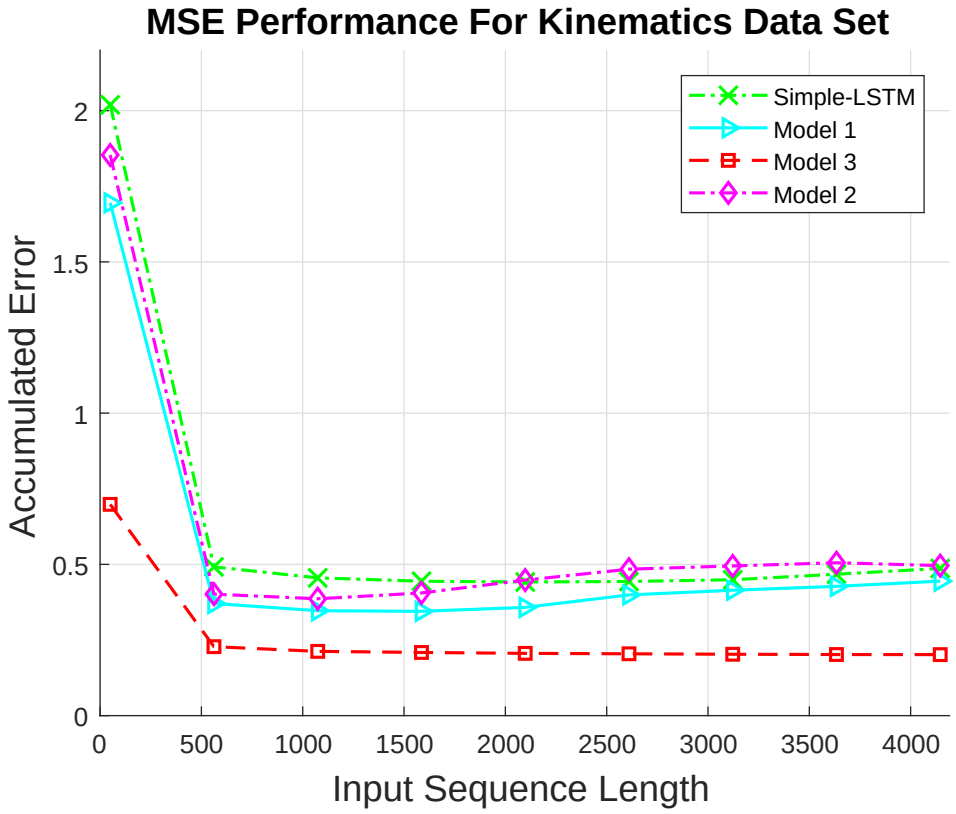

Figure 2.2: Accumulated error performance for the distance prediction performance of the robotic arm for the Kinematics data set using various Co-LSTM network models and Simple-LSTM networks (without the WMF).

information in the gating structure of the simple-LSTM networks, the overall performance increases. This covariance information is useful to train the Simple-LSTM networks and helps better in the overall regression performance. Out of all the models of the Co-LSTM networks, Model 3 outperforms all the other models and Simple-LSTM networks as shown in Fig. 2.2. Following this trend, Model 1 shows the second best regression performance. For the experiments, we keep the learning rate parameter in all the Co-LSTM network models and Simple-LSTM networks to be $\mu = 0.01$.

In Fig. 2.3, we compare the regression performance for the Elevators data set [23] for the proposed Co-LSTM models with simple LSTM networks. In the Elevators data set, we have an 18 dimensional input vector and based on this we need to predict the set of actions that are done by the F16 aircraft. We keep the

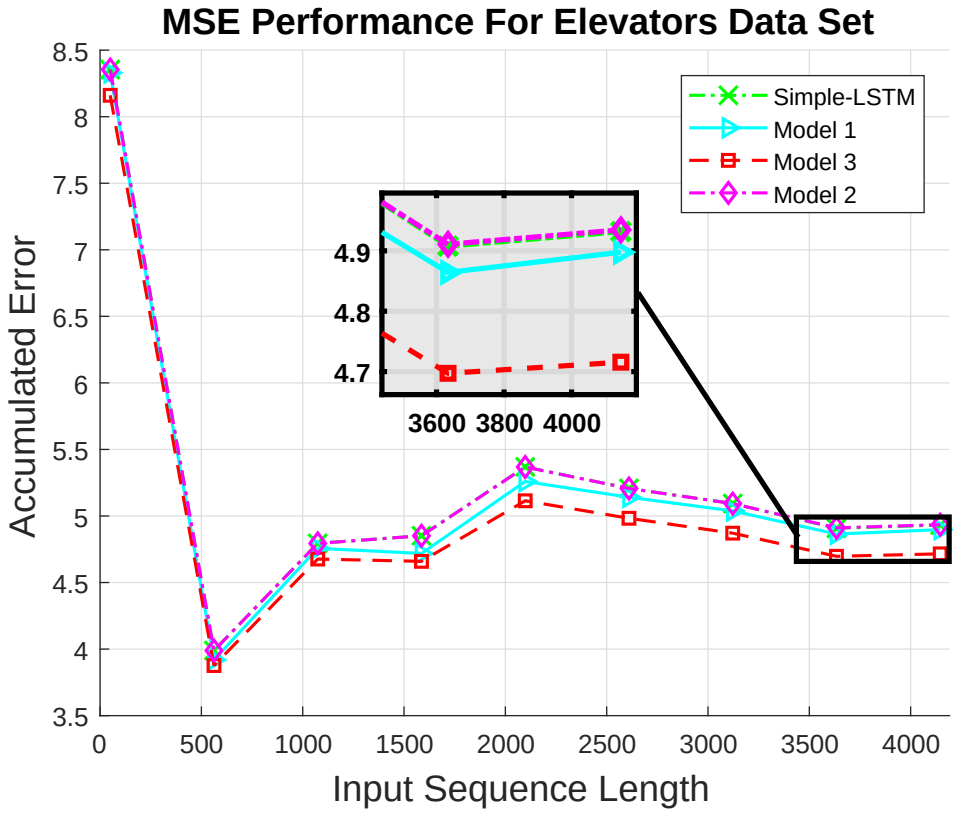

Figure 2.3: Accumulated error performance for the action prediction performance of an F16 aircraft for the Elevators data set using various Co-LSTM network models and Simple-LSTM networks (without the WMF).

learning rate to be $\mu = 0.01$ to be the same in all the set of experiments and for all the models. We observe in Fig. 2.3 that overall Model 3 of CO-LSTM networks shows the best performance as compared to all other models. Similarly, Model 1 shows the next best regression performance while Model 2 and Simple-LSTM networks show almost the same accumulated error trend.

In Table 2.3, we mention the steady state error performance of the Simple-LSTM and all the models of the Co-LSTM networks. As a whole, Model 3 has the lowest steady-state error in both of the data sets followed by Model 1. We then repeat the same set of experiments for the Simple and Co-LSTM networks with WMF. We observe from Table 2.3 that by employing the WMF approach, the performance is almost the same as that without WMF. Using the WMF approach drastically reduces the time and computational complexity while still

maintaining the accuracy of the system. In our experiments, we choose $rank = 2$ for all the data sets.

Table 2.2: Total number of network parameters for the Simple-LSTM and Co-LSTM (all models) networks with and without WMF for Kinematics, Elevators, Protein Tertiary and Puma8NH Data Sets.

Models / Data Sets	**Kinematcis**	**Elevators**	**Protein**	**Puma8NH**
Simple LSTM	220	2664	684	684
Co-LSTM (Model 1)	420	5256	1332	1332
Co-LSTM (Model 2)	220	2664	684	684
Co-LSTM (Model 3)	220	2664	684	684
WMF-Co-LSTM (Model 1)	340	1224	612	612
WMF-Co-LSTM (Model 2)	**180**	**648**	**324**	**324**
WMF-Co-LSTM (Model 3)	**180**	**648**	**324**	**324**

In Table 2.4, we list down the time (in seconds) taken for training the Simple and Co-LSTM networks with and without the WMF. Co-LSTM networks (Model 1 without WMF) takes almost 2 times more time to train the network as compared to the Simple-LSTM networks. With the inclusion of the WMF, the training time is significantly reduced as shown in Table 2.4.

In Table 2.2, we list down the actual number of parameters for the Simple and Co-LSTM networks with and without WMF for both of the data sets, i.e., Kinematics [22], Elevators [23], Protein Tertiary [24] and Puma8NH [25] data sets. From Table 2.3 and Table 2.2, we observe that Model 3 of the Co-LSTM (without the WMF) outperforms the Simple-LSTM networks in terms of performance and having the same number of parameters. But the training time for the Model 3 of the Co-LSTM networks (without the WMF) has slightly more training time as compared to that of the Simple-LSTM networks. Similarly, for the WMF version of the Model 3 of the Co-LSTM networks, the total number of

Table 2.3: Steady State Error Performance of Simple-LSTM and all the models of Co-LSTM networks with and without WMF.

Models / Data Sets	**Kinematics**	**Elevators**	**Protein**	**Puma8NH**
Simple LSTM	0.4866	0.4930	0.2569	0.1284
Co-LSTM (Model 1)	0.4460	4.898	0.2381	0.1111
Co-LSTM (Model 2)	0.4959	4.935	0.2679	0.1287
Co-LSTM (Model 3)	**0.2017**	**4.716**	**0.1764**	**0.1079**
WMF-Co-LSTM (Model 1)	0.4620	4.990	0.2578	0.1207
WMF-Co-LSTM (Model 2)	0.4913	4.9420	0.2699	0.1301
WMF-Co-LSTM (Model 3)	**0.2120**	**4.721**	**0.1780**	**0.1086**

parameters is drastically reduced while the performance almost remains the same with a very little variation. The major outcome of WMF-Co-LSTM (Model 3) networks is the decrease in the training time which is evident from the Table 2.4.

Remark 2.3: *Overall we found out that Model 3 depicts the best regression performance in all the data sets. Model 3 is the modified version of the Simple-LSTM networks. In Model 3, in the gating structure of the Simple-LSTM networks, we replace the weighted version of the input and the output with the weighted covariance version of the input and output respectively as shown in (2.10). Similarly, in the Model 1, we add additional weighted covariance input and output in the gating architecture of the Simple-LSTM networks as shown in (2.8).*

Remark 2.4: *Model 3 of the Co-LSTM networks shows the best performance among all of its models and Simple-LSTM networks. Model 3 has the same number of parameters as that of Simple-LSTM networks unlike Model 1 where the total number of parameters are approximately twice as compared to Simple-LSTM networks.*

We now consider the regression performances of the algorithms based on the GRU networks. In the previous sections, we use the LSTM architecture. Since our approach is generic, we also apply our approach to the recently introduced GRU architecture, which is described by the following set of equations:

$$\tilde{\boldsymbol{z}}_t = \sigma(\boldsymbol{W}^{(\tilde{z})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{z})}\boldsymbol{y}_{t-1}) \tag{2.24}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}_t + \boldsymbol{R}^{(r)}\boldsymbol{y}_{t-1}) \tag{2.25}$$

$$\tilde{\boldsymbol{y}}_t = g(\boldsymbol{W}^{(y)}\boldsymbol{x}_t + \boldsymbol{r}_t \odot (\boldsymbol{R}^{(y)}\boldsymbol{y}_{t-1})) \tag{2.26}$$

$$\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t \odot \tilde{\boldsymbol{z}}_t + \boldsymbol{y}_{t-1} \odot (\boldsymbol{1} - \tilde{\boldsymbol{z}}_t) \tag{2.27}$$

where $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output vector. The functions $g(\cdot)$ and $\sigma(\cdot)$ are set to the hyperbolic tangent and sigmoid functions, respectively. For the coefficient matrices, we have $\tilde{\boldsymbol{W}^{(z)}} \in \mathbb{R}^{m\times p}$, $\tilde{\boldsymbol{R}^{(z)}} \in \mathbb{R}^{m\times m}$, $\boldsymbol{W}^{(z)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(z)} \in \mathbb{R}^{m\times m}$, $\boldsymbol{W}^{(z)} \in \mathbb{R}^{m\times p}$ and $\boldsymbol{R}^{(z)} \in \mathbb{R}^{m\times m}$. Here, $\tilde{\boldsymbol{z}}_t$ and $\boldsymbol{r}_t$ are the update and reset gates, respectively. To obtain GRU-based algorithms, we directly replace the LSTM equations with the GRU equations and then apply our regression and training approaches. However, the GRU network lacks the output gate, which controls the amount of the incoming memory content. Furthermore, these networks differ in the location of the forget gates or the corresponding reset gates.

Remark 2.5: *We now perform the similar set of experiments (as conducted above using the LSTM networks) using the GRU networks and list down the performance. We investigate the steady state error performance, training time profile(in seconds) and the total number of parameters in Appendix B*

Table 2.4: Training times (in seconds) for the Simple-LSTM and all the models of the Co-LSTM networks with and without WMF.

Algorithms	**Time (Kinematics)**	**Time (Elevators)**	**Time (Protein)**	**Time (Puma8NH)**
Simple LSTM	63.27	768.14	430.08	450.18
Co-LSTM (Model 1)	139.14	1762.12	870.73	907.01
Co-LSTM (Model 2)	68.06	770.10	450.68	463.63
Co-LSTM (Model 3)	71.19	780.97	473.33	480.17
WMF-Co-LSTM (Model 1)	115.19	370.77	390.36	391.08
WMF-Co-LSTM (Model 2)	62.20	191.84	190.36	187.24
WMF-Co-LSTM (Model 3)	65.19	196.69	191.57	188.88

Chapter 3

Network Intrusion Detection Using the Co-LSTM Networks

In this chapter, we use the Co-LSTM networks proposed in Chapter 2 for a more practical application like network intrusion detection using network payload information. We illustrate the performance of the proposed algorithms using the intrusion detection evaluation data set (ISCX IDS 2012) which contains the source and destination payload information [26]. We perform network intrusion detection using both frameworks supervised and unsupervised framework. First, we experiment on a supervised Co-LSTM framework to perform binary network intrusion detection. We work with simple LSTM [12], Bi-LSTM [27] and its variants as explained in the subsequent sections. We then investigate the problem of unsupervised network intrusion detection where we develop a sequential LSTM decoder-encoder framework [28]. Finally, we illustrate the performance using the f1-score [29], AUC score and ROC curves [30].

3.1 Supervised Network Intrusion Detection

We have a network data sequence $\{\boldsymbol{X}_t\}_{t=1}^{n}$ of variable length, where $\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}, \ldots, \boldsymbol{x}_{t_{q_t}}]$ such that $q_t \in \mathbb{Z}^+$, $\boldsymbol{x}_{t_j} \in \mathbb{R}^d$, and $1 \leq j \leq q_t$. Each sequence is of variable length which is shown by q_t. Our aim is to perform network intrusion detection on variable length network data. Each network data sequence is composed of source and destination payloads. For a given network data sequence $\boldsymbol{X}_t$, we have

$$\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}^{(s)}, \boldsymbol{x}_{t_2}^{(s)}, \ldots, \boldsymbol{x}_{t_{v_t}}^{(s)}, \boldsymbol{x}_{t_1}^{(d)}, \boldsymbol{x}_{t_2}^{(d)}, \ldots, \boldsymbol{x}_{t_{u_t}}^{(d)}], \tag{3.1}$$

where $\boldsymbol{x}_{t_{v_t}}^{(s)} \in \mathbb{R}^d$ and $\boldsymbol{x}_{t_{u_t}}^{(d)} \in \mathbb{R}^d$ are the source and destinations payload vectors. In (3.8), we have $q_t = v_t + u_t$, where v_t and u_t depicts the variable length of the source and destination payloads. The main problem we face with the network data sequences is that we need to have a fixed length input sequence in order to identify the attack. We achieve fixed length data sequence by using the LSTM networks followed by a pooling layer as shown in the subsequent subsections. In this paper, we use the LSTM networks with one hidden layer defined as follows [12]:

$$\tilde{\boldsymbol{c}}_t = g\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{3.2}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\right) \tag{3.3}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}\right) \tag{3.4}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{3.5}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}\right) \tag{3.6}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot l(\boldsymbol{c}_t), \tag{3.7}$$

where $\boldsymbol{c}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^d$ is the input vector and $\boldsymbol{h}_t \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$ and $\boldsymbol{o}_t$ are the input, forget and output gates, respectively. The functions $g(\cdot)$ and $l(\cdot)$ apply to vectors pointwise and commonly set to $\tanh(\cdot)$. Similarly, the sigmoid function $\sigma(\cdot)$ applies pointwise to the vector elements and $\odot$ is the element by element multiplication of two vectors of the same size. The weight matrices are set to appropriate dimensions.

3.1.1 Co-LSTM Based Network Intrusion Detection

In this subsection, we develop the supervised framework for network intrusion detection using the Co-LSTM networks. We use all the models, i.e., (2.8), (2.9) and (2.10), instead of the Simple-LSTM networks to carry out the network intrusion detection. We pass the network data sequence to the Co-LSTM network (in all the models) whose output is then passed through the pooling layer. We perform three types of pooling, i.e., mean, max and last pooling. The pooling layer is then followed by a logistic regression layer for the classification purposes as shown in Fig. 3.1 and Fig. 3.2. We use two configurations on all the models of the Co-LSTM networks. In the first configuration, we have whole network data sequence (both source and destination payloads without break) while in the second configuration, we have a break between the source and destination payloads being passed to the Co-LSTM networks as shown in Fig. 3.1 and Fig. 3.2 respectively.

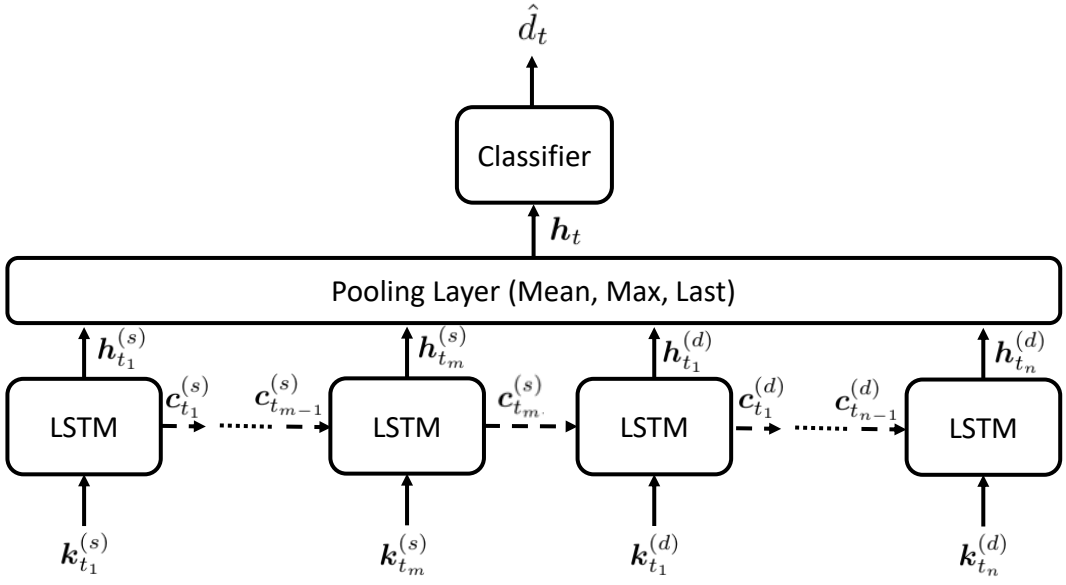

Figure 3.1: Detailed schematic diagram of LSTM based Configuration-1 network intrusion detection.

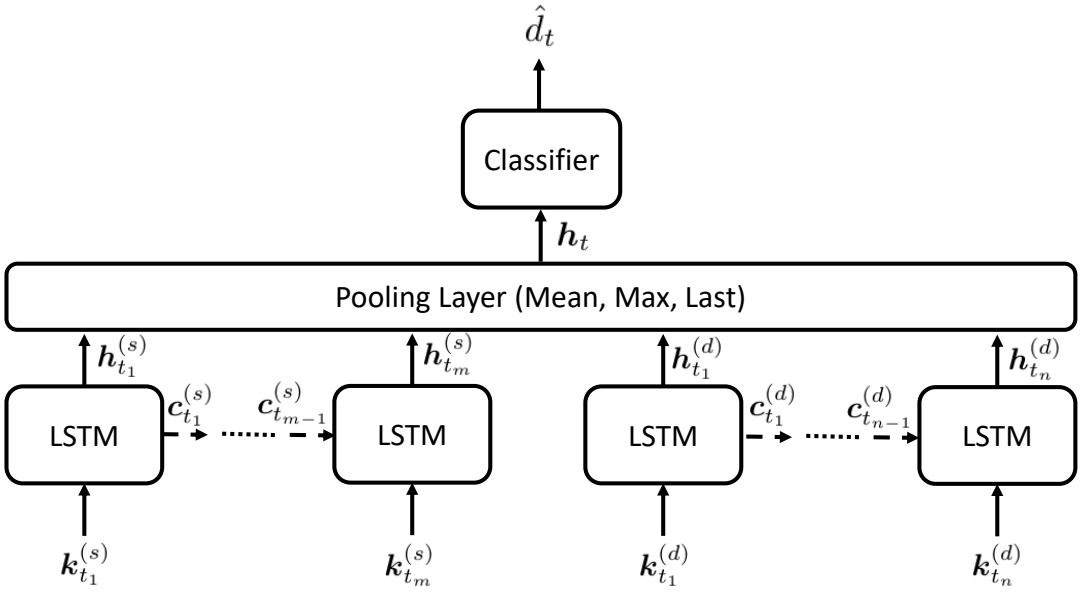

Figure 3.2: Detailed schematic diagram of LSTM based Configuration-2 network intrusion detection.

3.2 Unsupervised Network Intrusion Detection

We have a network data sequence $\{\boldsymbol{X}_t\}_{t=1}^{n}$ of variable length, where $\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}, \ldots, \boldsymbol{x}_{t_{q_t}}]$ such that $q_t \in \mathbb{Z}^+$, $\boldsymbol{x}_{t_j} \in \mathbb{R}^d$, and $1 \leq j \leq q_t$. Each sequence is of variable length which is shown by q_t. Our aim is to perform unsupervised network intrusion detection on variable length network data. Each network data sequence is composed of source and destination payloads. For a given network data sequence $\boldsymbol{X}_t$, we have

$$\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}^{(s)}, \boldsymbol{x}_{t_2}^{(s)}, \ldots, \boldsymbol{x}_{t_{v_t}}^{(s)}, \boldsymbol{x}_{t_1}^{(d)}, \boldsymbol{x}_{t_2}^{(d)}, \ldots, \boldsymbol{x}_{t_{u_t}}^{(d)}], \tag{3.8}$$

where $\boldsymbol{x}_{t_{v_t}}^{(s)} \in \mathbb{R}^d$ and $\boldsymbol{x}_{t_{u_t}}^{(d)} \in \mathbb{R}^d$ are the source and destinations payload vectors. In (3.8), we have $q_t = v_t + u_t$, where v_t and u_t depicts the variable length of the source and destination payloads.

We use the RNN to process the variable length input, such as $\boldsymbol{X}_t$, to extract the sequential natured information from the data. For each $\boldsymbol{X}_t$, the framework

for the generic RNN for the i^{th} column of $\boldsymbol{X}_t$ is given as follows [14], [15]:

$$\boldsymbol{h}_{t_i} = \kappa(\boldsymbol{W}\boldsymbol{x}_{t_i} + \boldsymbol{R}\boldsymbol{h}_{(t-1)_i}),$$

where $\boldsymbol{h}_{t_i} \in \mathbb{R}^m$ is the state vector and $\boldsymbol{x}_{t_i} \in \mathbb{R}^d$ is the input vector for $i = 1, \ldots, n_t$. The RNN coefficient weight matrices are $\boldsymbol{R} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{W} \in \mathbb{R}^{m \times d}$. The function $\kappa(\cdot)$ is commonly set to $\tanh(\cdot)$ and apply pointwise to vectors.

Now that we have the formulation of the RNN network, we extract the sequential information by driving each column of $\boldsymbol{X}_t$ to the encoder part of the RNN network. For each $\boldsymbol{X}_t$, the output is given by

$$\boldsymbol{h}_{t_i} = \kappa_{\phi}^{enc}(\boldsymbol{x}_{t_i}, \boldsymbol{h}_{(t-1)_i}), \tag{3.9}$$

where $\boldsymbol{h}_{t_i}$ is the output of the i^{th} RNN-encoder unit and ϕ is the parameter set of the RNN-encoder part. After whole of the sequence is passed through the RNN-encoder, we get $\{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}$. We then perform three types of pooling operation on $\{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}$, i.e., mean, max and last pooling. The mean, last and max pooling operations are computed as follows:

$$\boldsymbol{h}_i = \frac{\sum_{j=1}^{n_i} \boldsymbol{h}_{t_j}}{n_i} \tag{3.10}$$

$$\boldsymbol{h}_i = \boldsymbol{h}_{t,n_i} \tag{3.11}$$

$$\boldsymbol{h}_i = \max_j \{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}, \tag{3.12}$$

where j is the index for the number of rows of $\boldsymbol{h}_{t_i}$. After the pooling operation, we pass $\boldsymbol{h}_i$ to the RNN-decoder part which reconstructs the input as follows:

$$\hat{\boldsymbol{h}}_{t_i} = \kappa_{\psi}^{dec}(\boldsymbol{h}_i, \hat{\boldsymbol{h}}_{(t-1)_i}) \tag{3.13}$$

$$\hat{\boldsymbol{x}}_{t_i} = \rho(\hat{\boldsymbol{h}}_{t_i}), \tag{3.14}$$

where $\{\hat{\boldsymbol{x}}_{t_i}\}_{i=1}^{n_i}$ is the reconstructed input and ψ is the parameter set for RNN-decoder part. The function $\rho(\cdot)$ is commonly set to $\tanh(\cdot)$ and apply pointwise to vectors. After we retrieve the reconstructed input, we evaluate the mean square loss, i.e., $\sum_{i=1}^{n_i} ||\boldsymbol{x}_{t_i} - \hat{\boldsymbol{x}}_{t_i}||^2$ and update the corresponding LSTM-encoder and decoder parameters accordingly.

Having developed the unsupervised network intrusion detection framework for the RNN, we will use the Co-LSTM networks instead of the RNNs.

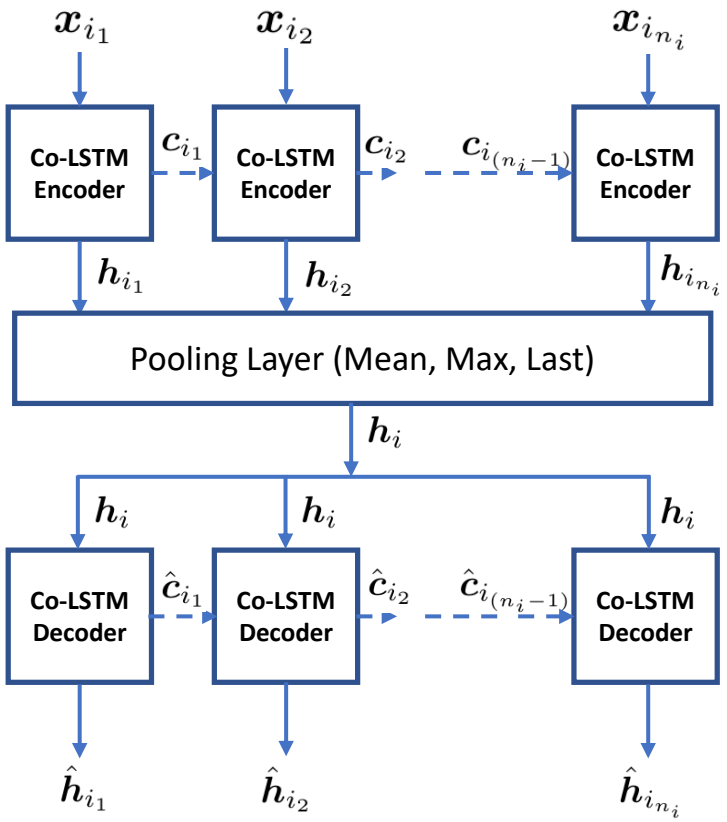

Figure 3.3: Detailed description of the Co-LSTM Sequential Autoencoder Model using the output of pooling layer, i.e., $\boldsymbol{h}_i$ as the input to all the stages of LSTM-decoder part.

Remark 3.1: *The RNN Autoencoder framework discussed in (3.9)-(3.14) also applies on all the models of the Co-LSTM neural networks defined in (2.8)-(2.10). The detailed description of the sequential Co-LSTM autoencoder framework is shown in Fig. 3.3. The encoder and decoder equations for the Co-LSTM network*

modifies as follows:

$$\boldsymbol{h}_{t_i} = \kappa_{\phi}^{enc}(\boldsymbol{x}_{t_i}, \boldsymbol{c}_{t_{i-1}}) \tag{3.15}$$

$$\hat{\boldsymbol{h}}_{t_i} = \kappa_{\psi}^{dec}(\boldsymbol{h}_i, \hat{\boldsymbol{h}}_{t_{i-1}}, \hat{\boldsymbol{c}}_{t_{i-1}}) \tag{3.16}$$

$$\hat{\boldsymbol{x}}_{t_i} = \rho(\hat{\boldsymbol{h}}_{t_i}), \tag{3.17}$$

where $\boldsymbol{h}_i$ is the Co-LSTM state vector obtained after pooling operation as mentioned in (3.10)-(3.12).

3.2.1 Error Function and Threshold

During the reconstruction phase in the sequential Co-LSTM autoencoder, there is an error associated with the reconstructed input. The reconstruction error for sequence $\boldsymbol{X}_i = [\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_{n_i}}]$ is given as follows:

$$Error(i) = \sum_{i=1}^{n_i} ||\boldsymbol{x}_{t_i} - \hat{\boldsymbol{x}}_{t_i}||^2, \tag{3.18}$$

where $Error(i)$ is the reconstruction error for sequence $\boldsymbol{X}_i$. Based on this error measure, we update the corresponding weights of encoder and decoder part of the Co-LSTM-autoencoder framework.

Remark 3.2: *For normal data sequences, the value of reconstruction error is less than the reconstruction error for anomalous data sequence. As a result, in order to classify the data as an anomaly, we assign a threshold value τ. The value of τ is critical, as it is directly related to the accuracy of the system. Table 1, shows the best achievable f1-score for a particular threshold value τ.*

In Fig. 3.4, we display the reconstruction error for the first 300 test samples after passing it through the Co-LSTM Encoder-Decoder framework. During the training process, the threshold value τ was selected as $\tau = 0.0092$. This threshold is shown by a red horizontal line in Fig. 3.4. The test samples whose error is greater than the threshold value, i.e., $Error_{test}(i) > \tau = Error_{test}(i) > 0.0092$ for $i = 1, \ldots, n_{test}$, will be considered as an anomaly and vice versa where n_{test}

is the length of the test data samples and $Error_{test}(i)$ is the reconstruction error for the i^th test sample.

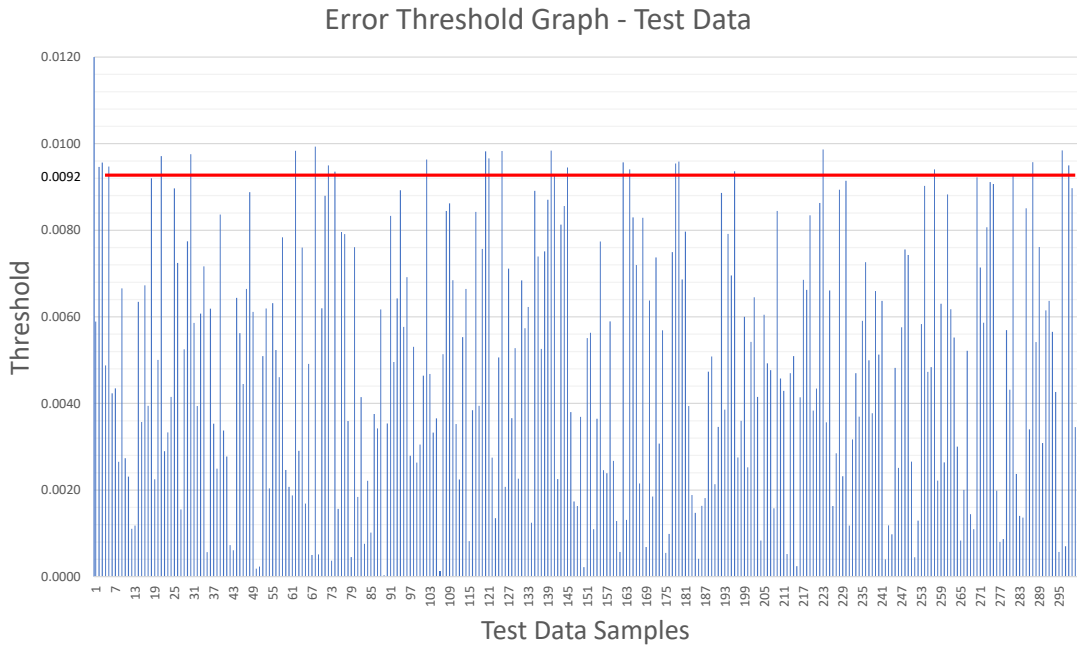

Figure 3.4: Threshold error graph

3.3 Experiments

In this section, we demonstrate the performance of our proposed algorithm using intrusion detection evaluation data set (ISCX IDS 2012) [26]. Network payloads are captured for seven days. There are around 1.8 millions of connections for FTP, SMTP, HTTP, SSH, IMAP, and POP3. Around five percent of connections are labelled as an anomaly. These anomalies come from a diverse set of multi-stage attacks. Some connections do not have packet payloads at the source or/and destination ports. Since packet payloads are used as input in our systems, we disregard the connections without payloads at both ports, regarding anomaly occurrence rate must remain almost the same after this operation.

Each network payload, captured at both source and destination ports, is regarded as sequential character-based input. The payloads are used in hexadecimal

Table 3.1: f1-scores and AUC-scores for all the algorithms with various pooling layers and configurations. Each block has three sub-row values in the style : mean, max and concatenate, and two sub-column values in the style : Configuration-1 and Configuration-2

Algorithms / Scores	**f1-score**	**AUC score**
LSTM	0.8617 , 0.8623	0.9666 , 0.9670
	0.8610 , 0.8599	0.9651 , 0.9644
	0.8627 , 0.8626	0.9693 , 0.9695
Bi-LSTM	0.8667 , 0.8574	0.9706 , 0.9700
	0.8409 , 0.8715	0.9512 , 0.9769
	0.8312 , 0.8008	0.8980 , 0.8722
Conv-LSTM	0.8678 , 0.8608	0.9671 , 0.9606
	0.8577 . 0.8578	0.9583 , 0.9853
	0.8701 , 0.8699	0.9722 , 0.9715
Conv-Bi-LSTM	0.8667 , 0.8555	0.9707 , 0.9642
	0.8674 , 0.8777	0.9508 , 0.9552
	0.8780 , 0.8699	0.9725 , 0.9691
Co-LSTM (Model-1)	0.8891 , 0.8876	0.9777 , 0.9771
	0.8765 , 08654	0.9623 , 0.9699
	0.8901 , 0.8808	0.9893 , 0.9893
Co-LSTM (Model-2)	0.8612 , 0.8666	0.9761 , 0.9566
	0.8705 , 0.8777	0.9777 , 0.9881
	0.8821 , 08810	0.9512 , 0.9555
Co-LSTM (Model-3)	0.8992 , 0.8975	0.9898 , 0.9870
	0.8883 , 0.8900	0.9782 , 0.9773
	0.9001 , 0.8997	0.9901 , 0.9883

format, so we have a total of 64 characters to be considered as our vocabulary. By using one hot encoding, characters are converted to numerical features resulting in 64-dimensional vectors.

We randomly split the data set into training and test sets with percentage 90 and 10, respectively. Among the training set, 20k of connections are chosen randomly to be used in our experiments. For all the splits, anomaly occurrence rate remains the same. As our training method, Adam [31] is employed with default parameters presented in the original work. The objective function is mean squared error. Batch size is chosen as 64.

Table 3.2: AUC-scores for the Random Forests (RF) classifier with various number of estimators and depth of the tree with unigrams modeling. In each tab, there are three values for the AUC-score. Top value is with mean pooling, center with max pooling and bottom value with concatenation pooling.

Depth / # of Estimators	**D=2**	**D=4**	**D=6**	**D=8**
N=10	0.8892	0.8928	0.8952	0.8952
	0.8912	0.8934	0.8937	0.8949
	0.8922	0.8937	0.8943	0.8942
N=20	0.8904	0.8936	0.8947	0.8957
	0.8905	0.8937	0.8948	0.8955
	0.8921	0.8950	0.8947	0.8942
N=30	0.8933	0.8941	0.8948	0.8958
	0.8915	0.8939	0.8943	0.8955
	0.8911	0.8949	0.8952	0.8956

3.3.1 Supervised Intrusion Detection Experiments

In this subsection, we perform extensive supervised intrusion detection experiments. First, we execute supervised network intrusion detection and record the f1-score [29] and AUC-score [30] using the Random Forests (RF) classifier [32] and Support Vector Machine (SVM) classifier [33] in Table 3.2 and Table 3.3 respectively. For both of the RF and SVM classifier, we first transform the network payload data to unigram model. We develop the unigram model for source and destination payloads separately. We then combine these two unigram models via mean, max and concatenation pooling.

For the RF classifier, we experiment with various number of estimators and depth of the tree. We perform each experiment using 5-fold cross validation and list down the best AUC score in Table 3.2.

Similarly, for the SVM classifier, we experiment with three different kernel functions, i.e., linear, RBF and polynomial. Here we experiment with different values of C, γ and d and mention their AUC scores in Table 3.3. Here again we deal with three types of pooling, i.e., mean, max and concatenation pooling, of source and destination unigram models.

Table 3.3: AUC scores for the SVM classifier with three different kernels, i.e., linear, RBF and polynomial, each with various combination values of its corresponding parameters. There are three AUC-score values in each tab where these values correspond to AUC-score values with mean, max and concatenate pooling values.

Kernels	**Parmeters**	**AUC Score**
Linear (C)	0.01	0.8769, 0.8761, 0.8790
	1	0.8952, 0.8952, 0.8965
	100	0.8557, 0.8434, 0.8100
RBF (C,γ)	(0.01,0.1)	0.8143, 0.8099, 0.8241
	(0.01,1)	0.8231, 0.8412, 0.8558
	(0.01,10)	0.7986, 0.7444, 0.6123
	(1,0.1)	0.8909, 0.8915, 0.8741
	(1,1)	0.8921, 0.8921, 0.8986
	(1,10)	0.8453, 0.7988, 0.7666
	(100,0.1)	0.7128, 0.7332, 0.7878
	(100,1)	0.8564, 0.7128, 0.8775
	(100,10)	0.8444, 0.8442, 0.8444
Polynomial (C,d)	(0.01,3)	0.8555, 0.8438, 0.8663
	(0.01,5)	0.8768, 0.8777, 0.8989
	(0.01,8)	0.8862, 0.8902, 0.8902
	(1,3)	0.8954, 0.8873, 0.8972
	(1,5)	0.8956, 0.8977, 0.8998
	(1,8)	0.8943, 0.8876, 0.8958
	(100,3)	0.8112, 0.7983, 0.7888
	(100,5)	0.7120, 0.6986, 0.7982
	(100,8)	0.7866, 0.7989, 0.8121

Now we proceed towards the LSTM networks and its variants. We mention the f1-score and AUC-score of the Co-LSTM networks (all three models) along with LSTM networks, Bidirectional-LSTM (Bi-LSTM) networks, Convolution-LSTM (Conv-LSTM) networks and Conv-Bi-LSTM networks [34]. The f1-scores and AUC-scores of the supervised algorithms is shown in Table 3.1. From Table 3.1, we observe that, Model-1 and Model-3 of the Co-LSTM networks outperforms all other supervised algorithms. Model-2 of the Co-LSTM networks performs almost the same as the Conv-Bi-LSTM networks but much better than the simple simple LSTM, Bi-LSTM and Conv-LSTM networks. All the LSTM networks and its variants perform much better than the RF and SVM classifiers. This is because the LSTM networks can capture the data dependency way much better than the

RF and SVM classifiers.

3.3.2 Unsupervised Intrusion Detection

In this subsection, we now shift the network intrusion detection towards the unsupervised framework. In this chapter, we specifically investigate the sequential autoencoder framework.

We implement Co-LSTM autoencoders having a unit size of 64 at both encoder and decoder layers. Sigmoid activation is used at the output layer. For all the experiments, 20k of data is split into training and test sets with size 16000 and 4000, respectively. The training set is further split into training and validation sets with 80/20 ratio. First, we compare various systems with the proposed Co-LSTM autoencoders as shown in Fig. 3.5. We also added more layers in the proposed algorithm, i.e., 2 layers each in encoder and decoder part, and call it Deep Auto-LSTM networks. For the case of the RNNs, the LSTM network, the GRU network, and the bidirectional LSTM networks all with one layer is applied separately with linear regression at the end. In addition, feed-forward neural networks with one layer are trained with the SGD algorithm [**?**]. We use the validation set to obtain the receiver operating characteristics (ROC) and choose the threshold τ corresponding to the best AUC score. Then, the test set is evaluated with the fixed threshold τ and f1 scores are evaluated as shown in Table. 5.1. The receiver operating characteristics with the corresponding AUC scores for the proposed sequential Co-LSTM autoencoders and other unsupervised algorithms is shown in Fig. 3.5.

We carry out all the experiments using 5-fold cross-validation. We show the ROC curve for one of the experiments for the Co-LSTM autoencoder with mean pooling in Fig. 3.6. The ROC and AUC scores for all the folds are shown in Fig. 3.6. The average AUC score is 0.96 with a standard deviation of 0.01.

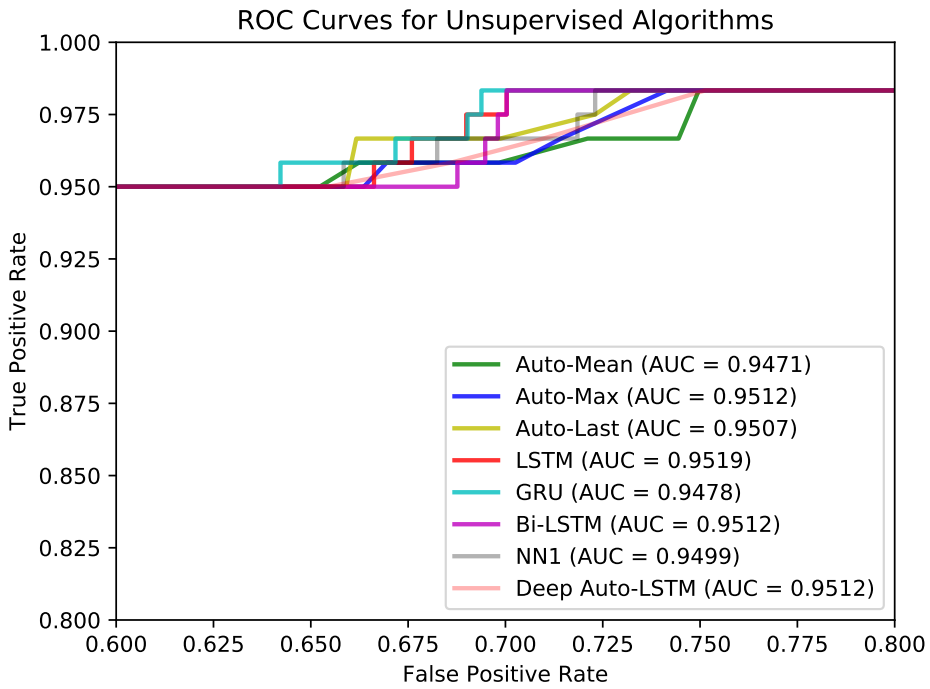

Figure 3.5: ROC curves for the proposed sequential Co-LSTM autoencoders along with several unsupervised algorithms for network intrusion detection.

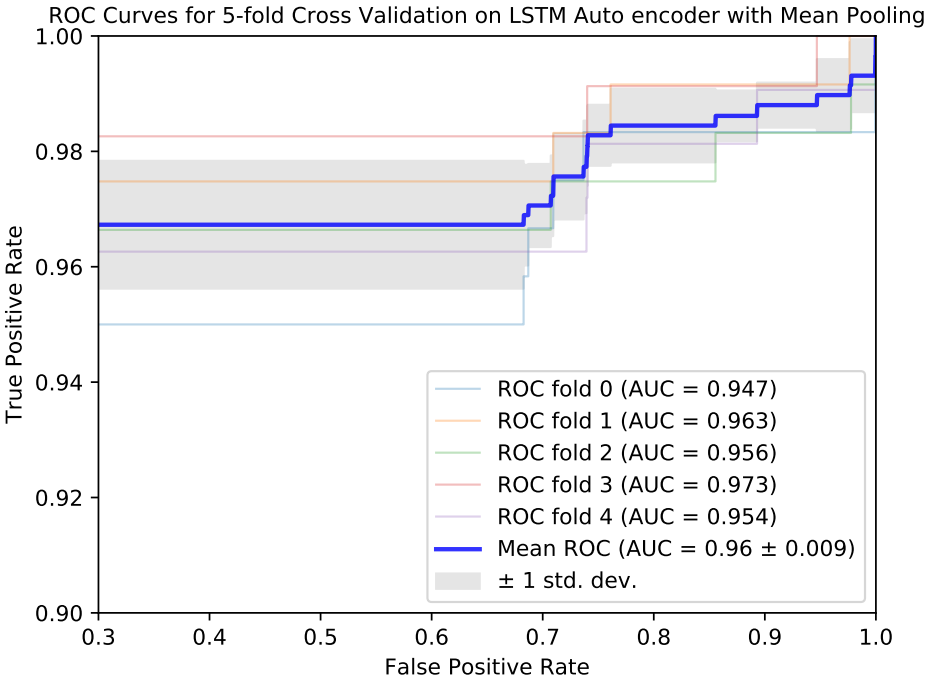

Figure 3.6: 5-fold cross-validation ROC curve for Co-LSTM Autoencoder with Mean Pooling.

Table 3.4: Performance Metrics Table showing the Threshold, f1-score and AUC score values for the sequential Co-LSTM Autoencoders and other Unsupervised Algorithms.

Unsupervised Learning Algorithms	**Threshold**	**f1-score**	**AUC Score**
LSTM	0.008	0.8409	**0.9519**
GRU	0.006	0.8102	0.9478
Bi-LSTM	0.008	0.8461	**0.9512**
NN1 - 1 layer	0.008	0.8352	0.9499
Co-LSTM Autoencoder with Last Pooling	0.092	0.8409	0.9507
Co-LSTM Autoencoder with Max Pooling	0.091	**0.8538**	**0.9512**
Co-LSTM Autoencoder with Mean Pooling	0.079	0.8072	0.9471
Deep Auto LSTM	0.076	**0.8538**	**0.9512**

Chapter 4

Low Complexity Boosted Binary Regression Tree-LSTM Neural Networks

In this chapter, we develop and implement regression-based LSTM networks using a tree-based architecture. The main purpose of using the tree-based structure is to add the boosting effect in the learning process. In [14], Tree-LSTM networks is introduced that efficiently captures the structural information. This structural information is of utmost importance in the sentiment analysis of the tweets and in analyzing the stock price data. The tree-based LSTM trees have an upper hand over the simple LSTM networks in terms of multiple dependency. This means that the LSTM gating vectors and memory cell updates depend on multiple LSTM units which are called as child LSTM nodes or units. Moreover, each parent LSTM node incorporates the combined effect of all the forget gates of its child LSTM nodes. There are two tree-based LSTM structures proposed in [19], [20] namely; Child-Sum Tree-LSTMs and N-ary Tree-LSTMs.

N-ary Tree LSTMs are introduced in [19], [20] that skips the sub-trees using their forget gates response that has a very little effect. The Tree-LSTM networks updates or manages its hidden state value based on the input and forget gate

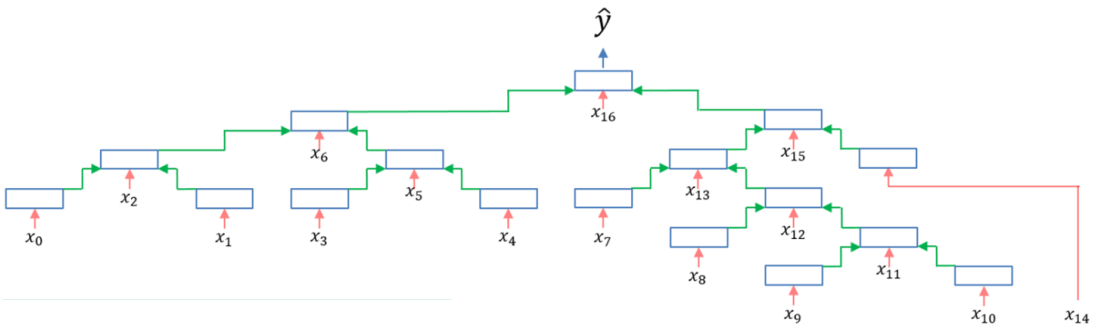

Figure 4.1: Schematic diagram of the basic Tree-LSTM network. The tree can either grow in a balanced manner or in an unbalanced manner.

response from its corresponding child [19], [20]. Whereas, in the simple LSTM networks, the hidden state at a current time is handled by the hidden state from the previous time step. Simple LSTM networks can be derived from the Tree-LSTM networks, i.e., they are a subset of Tree-LSTM networks.

In this chapter, we specifically modify the variant of the Tree-LSTM networks, i.e., N-ary Tree-LSTM networks. We introduce a boosted version of the Tree-LSTM networks. Instead of splitting the parent LSTM node into N child nodes, we split the parent LSTM node into two child LSTM nodes, i.e., binary splitting. We let the tree grow by binary splitting each node in a balanced manner thus obtaining a balanced tree in the end. The depth of the tree is denoted by α. We also introduce an input windowing factor that takes a fraction of input and insert into the child LSTM nodes according to each distinct depth of the tree. The information flow is from bottom to top and is in a hierarchical manner. While moving from bottom to top, we supply the previous depth child node forget gate response to the upper parent LSTM node. Each parent LSTM node takes in total two bottom child LSTM node forget gate response and process in such a way that it ignores the child LSTM node with a little effect. This process continues until we reach the ultimate parent LSTM node. The final decision is a combined effect that constitutes the forget gate response of all the child LSTM nodes. This depth factor and inclusion of input windowing factor to the child nodes are responsible for boosting the result of the network. We validate the performance of the proposed algorithms on real-life data sets. We use mean

square error (MSE) as performance criteria. We also take into account the total training time it requires to train the corresponding parameters.

4.1 Problem Description

For the regression framework, we receive input vectors $\{\boldsymbol{x}_t\}_{t\geq 1}$, $\boldsymbol{x}_t \in \mathbb{R}^n$ and $\{d_t\}_{t\geq 1}$, $d_t \in \mathbb{R}$ sequentially. Our aim is to estimate $\hat{d}_t = \boldsymbol{p}_t^T \boldsymbol{x}_t$, where $\boldsymbol{p}_t \in \mathbb{R}^n$. We then calculate the loss $\ell_t(d_t, \hat{d}_t)$ and update the corresponding parameters of the regression model. In this chapter, to estimate $\hat{d}_t$, besides using present sample $\boldsymbol{x}_t$, we use both present and past samples, i.e., $\{\boldsymbol{x}_{t-j}\}$ for $j = 1, \ldots, \nu$, where ν corresponds to the index of the past sample till which we want to use. We use recurrent neural networks (RNNs) to execute such a task. The basic framework of a simple RNN is given as follows [14]:

$$\boldsymbol{h}_t = \kappa(\boldsymbol{W}^{(h)}\boldsymbol{x}_t + \boldsymbol{R}^{(h)}\boldsymbol{h}_{t-1}) \tag{4.1}$$

$$\boldsymbol{y}_t = \psi(\boldsymbol{W}^{(y)}\boldsymbol{h}_t), \tag{4.2}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the input vector, $\boldsymbol{h}_t \in \mathbb{R}^m$ is the RNN state vector and $\boldsymbol{y}_t \in \mathbb{R}^m$ is the output vector. The functions $\kappa(\cdot)$ and $\psi(\cdot)$ are commonly set to $\tanh(\cdot)$ and applies pointwise to vectors. The coefficient weight matrices $\boldsymbol{W}^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)}$ are set to appropriate dimensions.

We specifically use the LSTM networks as a special case of the RNNs. The LSTM network with only one hidden layer is defined as follows [12]:

$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.3}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\right) \tag{4.4}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}\right) \tag{4.5}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{4.6}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}\right) \tag{4.7}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{4.8}$$

where $\boldsymbol{c}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^n$ is the input vector and $\boldsymbol{h}_t \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$ and $\boldsymbol{o}_t$ are the input, forget and output gates, respectively. The sigmoid function $\sigma(\cdot)$ and $\tanh(\cdot)$ applies pointwise to the vector elements. The weight matrices and bias vectors have the following dimensions, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m \times n}$, $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{b}^{(\cdot)} \in \mathbb{R}^m$.

4.2 Tree LSTM Networks

4.2.1 Child-Sum Tree-LSTM

Let $Ch(j) = \{ch_{j1}, ch_{j2}, \ldots, ch_{j\rho_j}\}$ denotes the set of children at the node j where ρ_j denotes the fixed or variable number of child node. The set of equations for the Child-Sum Tree-LSTM for node j where $k \in Ch(j)$ are written as follows [19]:

$$\tilde{\boldsymbol{h}_j} = \sum_{k \in Ch(j)} \boldsymbol{h}_k \tag{4.9}$$

$$\tilde{\boldsymbol{c}}_j = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_j + \boldsymbol{R}^{(\tilde{c})}\tilde{\boldsymbol{h}_j} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.10}$$

$$\boldsymbol{i}_j = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_j + \boldsymbol{R}^{(i)}\tilde{\boldsymbol{h}_j} + \boldsymbol{b}^{(i)}\right) \tag{4.11}$$

$$\boldsymbol{f}_{jk} = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{h}_k + \boldsymbol{b}^{(f)}\right) \tag{4.12}$$

$$\boldsymbol{c}_j = \boldsymbol{i}_j \odot \tilde{\boldsymbol{c}}_j + \sum_{k \in Ch(j)} \boldsymbol{f}_{jk} \odot \boldsymbol{c}_k \tag{4.13}$$

$$\boldsymbol{o}_j = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_j + \boldsymbol{R}^{(o)}\tilde{\boldsymbol{h}_j} + \boldsymbol{b}^{(o)}\right) \tag{4.14}$$

$$\boldsymbol{h}_j = \boldsymbol{o}_j \odot \tanh(\boldsymbol{c}_j). \tag{4.15}$$

4.2.2 N-ary Tree-LSTM

Let N be the branching factor at most where the children are indexed and ordered, then for the node j, the set of equations for the $N-$ary Tree LSTM for $k =$

$1, \ldots, N$ where N is the total number of child nodes, are given as follows [19], [20]:

$$\tilde{\boldsymbol{c}}_j = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{R}_l^{(\tilde{c})}\boldsymbol{h}_{jl} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.16}$$

$$\boldsymbol{i}_j = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{R}_l^{(i)}\boldsymbol{h}_{jl} + \boldsymbol{b}^{(i)}\right) \tag{4.17}$$

$$\boldsymbol{f}_{jk} = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \sum_{l=1}^{N} \boldsymbol{R}_{kl}^{(f)}\boldsymbol{h}_{jl} + \boldsymbol{b}^{(f)}\right) \tag{4.18}$$

$$\boldsymbol{c}_j = \boldsymbol{i}_j \odot \tilde{\boldsymbol{c}}_j + \sum_{l=1}^{N} \boldsymbol{f}_{jl} \odot \boldsymbol{c}_{jl} \tag{4.19}$$

$$\boldsymbol{o}_j = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{R}_l^{(o)}\boldsymbol{h}_{jl} + \boldsymbol{b}^{(o)}\right) \tag{4.20}$$

$$\boldsymbol{h}_j = \boldsymbol{o}_j \odot \tanh(\boldsymbol{c}_j), \tag{4.21}$$

where $\boldsymbol{h}_{jl}$ and $\boldsymbol{c}_{jl}$ is the output vector and state vector of the k^{th} child respectively. The detailed schematic diagram of this configuration is shown in Fig. 4.2. In this configuration, the forget gate of all the child LSTM nodes has an impact on the parent LSTM network. Based on this configuration, we modify the architecture and introduce boosted binary tree-LSTM (BBT-LSTM) in the following subsection.

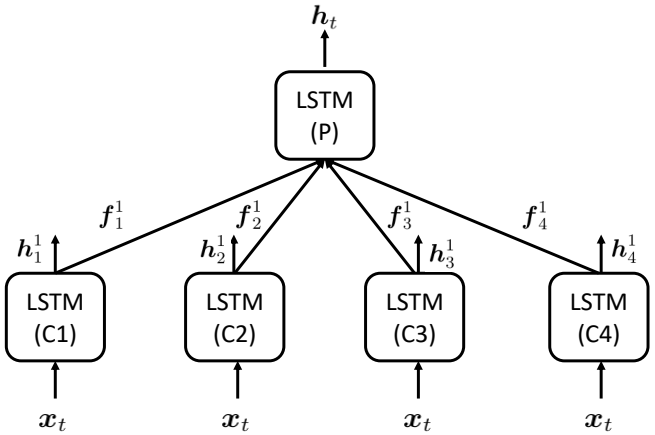

Figure 4.2: Detailed schematic diagram of the N-ary Tree-LSTM network with $N = 4$.

4.3 Boosted Binary Tree LSTM (BBT-LSTM) Networks

In this subsection, we introduce and formulate the mathematical model for layered tree LSTM structure and boost the overall effect of the input. In this architecture, we have a layered tree structure where each node is an LSTM network. Unlike $N-$ary Tree LSTM network, here we have binary splitting at each node of the layer instead of N splittings in only one layer. Let α define the index of depth, then the BBT-LSTM defined for the j^{th} node at a depth of γ of the tree is as follows:

$$\tilde{\boldsymbol{c}}_j^{\gamma} = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_l^{(\tilde{c})}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.22}$$

$$\boldsymbol{i}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_l^{(i)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(i)}\right) \tag{4.23}$$

$$\boldsymbol{f}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_l^{(f)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(f)}\right) \tag{4.24}$$

$$\boldsymbol{c}_j^{\gamma} = \boldsymbol{i}_j^{\gamma} \odot \tilde{\boldsymbol{c}}_j^{\gamma} + \sum_{l=1}^{2}\boldsymbol{f}_{jl}^{\gamma-} \odot \boldsymbol{c}_{jl}^{\gamma-} \tag{4.25}$$

$$\boldsymbol{o}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_l^{(o)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(o)}\right) \tag{4.26}$$

$$\boldsymbol{h}_j^{\gamma} = \boldsymbol{o}_j^{\gamma} \odot \tanh(\boldsymbol{c}_j^{\gamma}), \tag{4.27}$$

where $\boldsymbol{h}_j^{\gamma}$ is the output vector for the β LSTM node at a depth of γ. In the above equations, $\{\boldsymbol{h}_{jl}^{-}\}_{l=1}^{2} = \{\boldsymbol{h}_{j1}^{-}, \boldsymbol{h}_{j2}^{-}\}$ are the left and right child LSTM output vectors corresponding to the parent LSTM node at γ depth. The detailed schematic diagram for the BBT-LSTM network is shown in Fig. 4.3 specifically for depth $\alpha = 2$ and $N = 4$.

Remark 4.1: *Increasing the value of depth α increases the overall boosting effect and hence results in reducing the overall accumulated MSE. But this reduction in MSE comes at the cost of more time needed for training and processing. Overall, the computational complexity increases exponentially as the depth increase. Also,*

for the greater value of α, the framework may overfit on the data and hence will result in poor performance on the test data. So a proper trade-off must be established considering the size of the data set and the required result to be achieved.

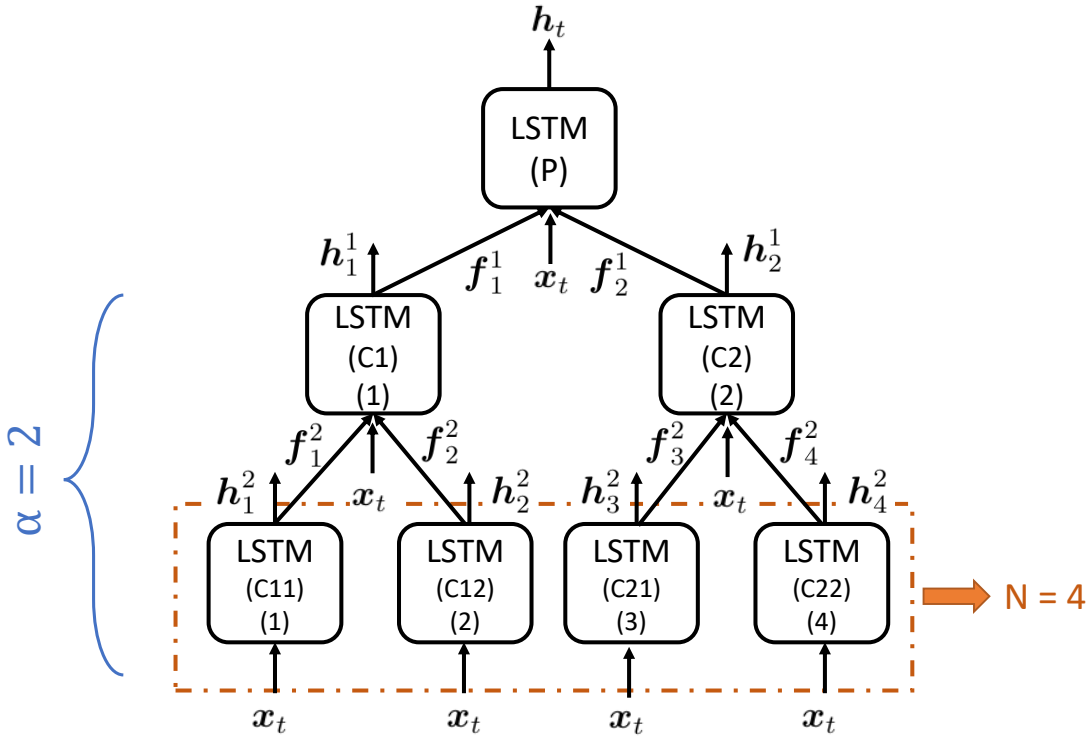

Figure 4.3: Detailed schematic diagram of the BBT-LSTM network with depth $\alpha = 2$ and $N = 4$.

The computational complexity of the simple LSTM, $N-$ary tree LSTM and the BBT-LSTM networks is shown in Table 4.1. In Table 4.1, n is the dimension of the input vector, m is the dimension of the state vector and N is the total number of child nodes in the tree based LSTM networks. We observe from Table 4.1 that the complexity increases exponentially in the tree based LSTM networks besides yielding the boosting effect. So considering the performance restraints, this exponential complexity is a very big deal and we intend to decrease the computational complexity in the upcoming subsections.

Table 4.1: Computational Complexity of the simple LSTM, $N-$ary Tree LSTM and BBT-LSTM networks. N represents the total number of child nodes in tree LSTM networks, n and m are the dimensions of the the input and state vector of the LSTM networks.

Algorithm	**Computational Complexity**
LSTM	$4(mn + mm + m)$
Tree LSTM	$4(mn + mm + m)(N + 1)$
BBT-LSTM	$4(mn + mm + m)(2^{\alpha+1} - 1)$

4.3.1 WMF Based BBT-LSTM Networks

In this subsection, we use the WMF on the BBT-LSTM networks. We make use of the idea from [17] and apply it to the BBT-LSTM networks. Let $\boldsymbol{A}$ be a rectangular matrix such that $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, then by using WMF we can write $\boldsymbol{A} \approx \boldsymbol{A}_1\boldsymbol{A}_2$ where $\boldsymbol{A}_1 \in \mathbb{R}^{m\times p}$ and $\boldsymbol{A}_2 \in \mathbb{R}^{p\times n}$ such that $p << \min(m, n)$. Following the WMF procedure, we can write the WMF based BBT-LSTM (WMF-BBT-LSTM) networks with coefficient weight matrices, i.e., $\boldsymbol{W}^{(\cdot)} \approx \boldsymbol{W}_1^{(\cdot)}\boldsymbol{W}_2^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)} \approx \boldsymbol{R}_1^{(\cdot)}\boldsymbol{R}_2^{(\cdot)}$ as follows:

$$\tilde{\boldsymbol{c}}_j^{\gamma} = \tanh\left(\boldsymbol{W}_1^{(\tilde{c})}\boldsymbol{W}_2^{(\tilde{c})}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_{l1}^{(\tilde{c})}\boldsymbol{R}_{l2}^{(\tilde{c})}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.28}$$

$$\boldsymbol{i}_j^{\gamma} = \sigma\left(\boldsymbol{W}_1^{(i)}\boldsymbol{W}_2^{(i)}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_{l1}^{(i)}\boldsymbol{R}_{l2}^{(i)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(i)}\right) \tag{4.29}$$

$$\boldsymbol{f}_j^{\gamma} = \sigma\left(\boldsymbol{W}_1^{(f)}\boldsymbol{W}_2^{(f)}\boldsymbol{x}_t + \sum_{l=1}^{2}\boldsymbol{R}_{l_1}^{(f)}\boldsymbol{R}_{l_2}^{(f)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(f)}\right) \tag{4.30}$$

$$\boldsymbol{c}_j^{\gamma} = \boldsymbol{i}_j^{\gamma} \odot \tilde{\boldsymbol{c}}_j^{\gamma} + \sum_{l=1}^{2}\boldsymbol{f}_{jl}^{\gamma-} \odot \boldsymbol{c}_{jl}^{\gamma-} \tag{4.31}$$

$$\boldsymbol{o}_j^{\gamma} = \sigma\left(\boldsymbol{W}_1^{(o)}\boldsymbol{W}_2^{(o)}\boldsymbol{x}_j + \sum_{l=1}^{2}\boldsymbol{R}_{l1}^{(o)}\boldsymbol{R}_{l2}^{(o)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(o)}\right) \tag{4.32}$$

$$\boldsymbol{h}_j^{\gamma} = \boldsymbol{o}_j^{\gamma} \odot \tanh(\boldsymbol{c}_j^{\gamma}). \tag{4.33}$$

Proposition 4.1: *The total number of parameters for the WMF-BBT-LSTM networks are less than the total number of parameters of the BBT-LSTM networks.*

Proof: Let $P_{(WMF-BBT-LSTM)}$ and $P_{(BBT-LSTM)}$ be the total number of parameters of the BBT-LSTM networks with and without the WMF respectively. We have

$$\begin{aligned}\frac{P_{(WMF-BBT-LSTM)}}{P_{(BBT-LSTM)}} &= \frac{4(mp+pn+2mq+m)(2^{\alpha+1}-1)}{4(mn+mm+m)(2^{\alpha+1}-1)} \\ &= \frac{p(m+n+2mq)+m}{m(m+n)+m} \\ &\approx \frac{p(m+n+2mq)}{m(m+n)} \\ &\approx \frac{p}{m}+\frac{2q}{m+n} \end{aligned} \tag{4.34}$$

Since $p << \min(m,n)$ and $q << m$, (4.34) implies that $P_{(WMF-BBT-LSTM)} < P_{(BBT-LSTM)}$.

□

Proposition 4.2: *The total number of parameters for the WMF-BBT-LSTM networks are less than the total number of parameters of the N-ary Tree networks.*

Proof: Let $P_{(WMF-BBT-LSTM)}$ and $P_{(N-ary)}$ be the total number of parameters of the BBT-LSTM networks with WMF and N-ary Tree networks. We have

$$\begin{aligned}\frac{P_{(WMF-BBT-LSTM)}}{P_{(N-ary)}} &= \frac{4(mp+pn+2mq+m)(2^{\alpha+1}-1)}{4(mn+mm+m)(N+1)} \\ &\quad \text{Since } 2^{\alpha+1}-1 = N+\sum_{\beta=0}^{\alpha-1}2^{\beta} \\ &= \frac{\big(p(m+n+2mq)+m\big)\big(N+\sum_{\beta=0}^{\alpha-1}2^{\beta}\big)}{\big(m(m+n)+m\big)\big(N+1\big)} \\ &\approx \left(\frac{p}{m}+\frac{2q}{m+n}\right)\left(\frac{N+\sum_{\beta=0}^{\alpha-1}2^{\beta}}{N+1}\right) \end{aligned} \tag{4.35}$$

Since $p << \min(m,n)$ and $q << m$, the first expression $\left(\frac{p}{m}+\frac{2q}{m+n}\right) << 1$. For the second expression in (4.35), i.e., $\left(\frac{N+\sum_{\beta=0}^{\alpha-1}2^{\beta}}{N+1}\right) > 1$ but the

overall combined effect of both the expressions in (4.35) implies that $\left(\frac{p}{m} + \frac{2q}{m+n}\right)\left(\frac{N+\sum_{\beta=0}^{\alpha-1} 2^{\beta}}{N+1}\right) < 1$ which results in $P_{(WMF-BBT-LSTM)} < P_{(N-ary)}$.

□

4.3.2 ef-operator Based BBT-LSTM Networks

In this subsection, we replace the regular multiplication operator involved in the matrix-vector and vector-vector with the energy efficient operator known as ef-operator [35], [36]. Let $\boldsymbol{r}$ and $\boldsymbol{s} \in \mathbb{R}^p$, then the ef-operation on $\boldsymbol{r}$ and $\boldsymbol{s}$ is defined as follows:

$$\boldsymbol{r} \diamond \boldsymbol{s} := \sum_{i=1}^{p} \text{sign}(r_i \times s_i)(|r_i| + |s_i|). \tag{4.36}$$

The above equation (4.36) can also be written as follows [35], [36]:

$$\boldsymbol{r} \diamond \boldsymbol{s} := \sum_{i=1}^{p} \text{sign}(r_i)s_i + \text{sign}(s_i)r_i. \tag{4.37}$$

Remark 4.2: *From (4.36) and (4.37)*, it is obvious that the ef-operator uses simple addition and most importantly sign multiplications instead of costly regular multiplications.

We now replace the simple multiplication operator with the ef-operator for the RNNs defined in (4.1) as follows:

$$\boldsymbol{h}_t = \gamma\big(\boldsymbol{a}_h \odot (\boldsymbol{W}^{(h)} \diamond \boldsymbol{x}_t) + \boldsymbol{b}_h \odot (\boldsymbol{R}^{(h)} \diamond \boldsymbol{h}_{t-1})\big), \tag{4.38}$$

where $\boldsymbol{a}_h$ and $\boldsymbol{b}_h$ are the scaling coefficients as mentioned in [ref]. The terms $\boldsymbol{W}^{(h)} \diamond \boldsymbol{x}_t$ and $\boldsymbol{R}^{(h)} \diamond \boldsymbol{h}_{t-1}$ mentioned in (4.38) are written as follows:

$$\boldsymbol{W}^{(h)} \diamond \boldsymbol{x}_t = [\boldsymbol{w}_1^{(h)} \diamond \boldsymbol{x}_t \boldsymbol{w}_2^{(h)} \diamond \boldsymbol{x}_t \ldots \boldsymbol{w}_m^{(h)} \diamond \boldsymbol{x}_t]^T,$$

where

$$\boldsymbol{w}_i^{(h)} \diamond \boldsymbol{x}_t = \sum_{j=1}^{n} \text{sign}(x_{t,j}) w_{ij}^{(h)} + \text{sign}(w_{ij}^{(h)}) x_{t,j},$$

where $\boldsymbol{w}_i^{(h)}$ is the transpose of the i^{th} row of $\boldsymbol{w}^{(h)}$ and $x_{t,j}$ is the j^{th} element of $\boldsymbol{x}_t$. Similarly $\boldsymbol{R}^{(h)} \diamond \boldsymbol{h}_{t-1}$ can be written as follows:

$$\boldsymbol{R}^{(h)} \diamond \boldsymbol{h}_{t-1} = [\boldsymbol{r}_1^{(h)} \diamond \boldsymbol{h}_{t-1} \boldsymbol{r}_2^{(h)} \diamond \boldsymbol{h}_{t-1} \dots \boldsymbol{r}_m^{(h)} \diamond \boldsymbol{h}_{t-1}]^T,$$

where

$$\boldsymbol{r}_i^{(h)} \diamond \boldsymbol{h}_{t-1} = \sum_{j=1}^{m} \text{sign}(h_{t-1,j}) r_{ij}^{(h)} + \text{sign}(r_{ij}^{(h)}) h_{t-1,j}.$$

Following the above procedure, we now apply the ef-operator to the BBT-LSTM networks. By using the ef-operator defined in (4.36) and (4.37), we can write (4.22)-(4.27) as follows:

$$\tilde{\boldsymbol{c}}_j^{\gamma} = \tanh\left(\boldsymbol{a}_{\tilde{c}} \odot \left(\boldsymbol{W}^{(\tilde{c})} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_{\tilde{c}}} \odot \left(\boldsymbol{R}_l^{(\tilde{c})} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.39}$$

$$\boldsymbol{i}_j^{\gamma} = \sigma\left(\boldsymbol{a}_i \odot \left(\boldsymbol{W}^{(i)} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_i} \odot \left(\boldsymbol{R}_l^{(i)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(i)}\right) \tag{4.40}$$

$$\boldsymbol{f}_j^{\gamma} = \sigma\left(\boldsymbol{a}_f \odot \left(\boldsymbol{W}^{(f)} \diamond \boldsymbol{x}_t\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_f} \odot \left(\boldsymbol{R}_l^{(f)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(f)}\right) \tag{4.41}$$

$$\boldsymbol{c}_j^{\gamma} = \boldsymbol{i}_j^{\gamma} \diamond \tilde{\boldsymbol{c}}_j^{\gamma} + \sum_{l=1}^{2} \boldsymbol{f}_{jl}^{\gamma-} \diamond \boldsymbol{c}_{jl}^{\gamma-} \tag{4.42}$$

$$\boldsymbol{o}_j^{\gamma} = \sigma\left(\boldsymbol{a}_o \odot \left(\boldsymbol{W}^{(o)} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_o} \odot \left(\boldsymbol{R}_l^{(o)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(o)}\right) \tag{4.43}$$

$$\boldsymbol{h}_j^{\gamma} = \boldsymbol{o}_j^{\gamma} \diamond \tanh(\boldsymbol{c}_j^{\gamma}). \tag{4.44}$$

Remark 4.3: *The sole purpose of using the ef-operator instead of the regular multiplication operator is to reduce the time complexity of the BBT-LSTM networks. Because of the tree architecture, the training and processing requires time and in order to alleviate the time complexity, ef-operator significantly reduces the complexity while still maintaining the performance results and accuracy.*

4.3.3 ef-WMF based BBT-LSTM Networks

In this subsection, we focus on both reducing the time and computational complexity of the BBT-LSTM networks. For this purpose, we combine the techniques as discussed in 4.3.1 and 4.3.2. Together applying the WMF on the network weight matrices of the BBT-LSTM networks along with replacing the regular multiplication operator with the energy efficient operator, i.e., ef-operator, the BBT-LSTM networks can be written as follows:

$$\tilde{\boldsymbol{c}}_j^{\gamma} = \tanh\left(\boldsymbol{a}_{\tilde{c}} \odot \left(\boldsymbol{W}_1^{(\tilde{c})}\boldsymbol{W}_2^{(\tilde{c})} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_{\tilde{c}}} \odot \left(\boldsymbol{R}_{l1}^{(\tilde{c})}\boldsymbol{R}_{l2}^{(\tilde{c})} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.45}$$

$$\boldsymbol{i}_j^{\gamma} = \sigma\left(\boldsymbol{a}_i \odot \left(\boldsymbol{W}_1^{(i)}\boldsymbol{W}_2^{(i)} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_i} \odot \left(\boldsymbol{R}_{l1}^{(i)}\boldsymbol{R}_{l2}^{(i)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(i)}\right) \tag{4.46}$$

$$\boldsymbol{f}_j^{\gamma} = \sigma\left(\boldsymbol{a}_f \odot \left(\boldsymbol{W}_1^{(f)}\boldsymbol{W}_2^{(f)} \diamond \boldsymbol{x}_t\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_f} \odot \left(\boldsymbol{R}_{l_1}^{(f)}\boldsymbol{R}_{l_2}^{(f)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(f)}\right) \tag{4.47}$$

$$\boldsymbol{c}_j^{\gamma} = \boldsymbol{i}_j^{\gamma} \diamond \tilde{\boldsymbol{c}}_j^{\gamma} + \sum_{l=1}^{2} \boldsymbol{f}_{jl}^{\gamma-} \diamond \boldsymbol{c}_{jl}^{\gamma-} \tag{4.48}$$

$$\boldsymbol{o}_j^{\gamma} = \sigma\left(\boldsymbol{a}_o \odot \left(\boldsymbol{W}_1^{(o)}\boldsymbol{W}_2^{(o)} \diamond \boldsymbol{x}_j\right) + \sum_{l=1}^{2} \boldsymbol{b}_{l_o} \odot \left(\boldsymbol{R}_{l1}^{(o)}\boldsymbol{R}_{l2}^{(o)} \diamond \boldsymbol{h}_{jl}^{-}\right) + \boldsymbol{b}^{(o)}\right) \tag{4.49}$$

$$\boldsymbol{h}_j^{\gamma} = \boldsymbol{o}_j^{\gamma} \diamond \tanh(\boldsymbol{c}_j^{\gamma}). \tag{4.50}$$

Remark 4.4: *Replacing the multiplication operator with the ef-operator reduces the time complexity but adds a little to the computational complexity because of the scaling coefficients, i.e.,* $\boldsymbol{a}_{(\cdot)}$ *and* $\boldsymbol{b}_{(\cdot)}$ *[35], [36]. In order to counteract the effect of increasing computational complexity, we then apply the WMF on the network weight matrices that significantly reduces the computational complexity. With the reduces time and computational complexity, the overall accuracy of the model almost remains unchanged.*

4.4 Slicing Operation based BBT-LSTM Networks

Besides WMF, in this section, we propose another way to further reduce the computational complexity of the BBT-LSTM networks. We introduce a slicing operation on the coefficient network weight matrices. Let α be the index of depth of the BBT-LSTM network and ω be the window length. For the BBT-LSTM network with coefficient network weight matrices, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$, with depth α and let $\lceil (\cdot) \rceil$ be the ceiling function, we define the slicing operation as

- $\boldsymbol{W}^{(\cdot)}$: select and retain a random set of $n - \lceil \frac{\omega}{2^\alpha} \rceil$ entries in each m rows of $\boldsymbol{W}^{(\cdot)}$ and setting remaining $\lceil \frac{\omega}{2^\alpha} \rceil$ entries in each m rows to be 0
- $\boldsymbol{R}^{(\cdot)}$: select and retain a random set of $m - \lceil \frac{\omega}{2^\alpha} \rceil$ entries in each m rows of $\boldsymbol{R}^{(\cdot)}$ and setting remaining $\lceil \frac{\omega}{2^\alpha} \rceil$ entries in each m rows to be 0

Based on the slicing operation defined above, the Sliced-BBT-LSTM networks can be written as follows:

$$\tilde{\boldsymbol{c}}_j^{\gamma} = \tanh\left(\boldsymbol{W}^{\gamma_\alpha^\omega(\tilde{c})}\boldsymbol{x}_j + \sum_{l=1}^{2} \boldsymbol{R}_l^{\gamma_\alpha^\omega(\tilde{c})}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{4.51}$$

$$\boldsymbol{i}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{\gamma_\alpha^\omega(i)}\boldsymbol{x}_j + \sum_{l=1}^{2} \boldsymbol{R}_l^{\gamma_\alpha^\omega(i)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(i)}\right) \tag{4.52}$$

$$\boldsymbol{f}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{\gamma_\alpha^\omega(f)}\boldsymbol{x}_t + \sum_{l=1}^{2} \boldsymbol{R}_l^{\gamma_\alpha^\omega(f)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(f)}\right) \tag{4.53}$$

$$\boldsymbol{c}_j^{\gamma} = \boldsymbol{i}_j^{\gamma} \odot \tilde{\boldsymbol{c}}_j^{\gamma} + \sum_{l=1}^{2} \boldsymbol{f}_{jl}^{\gamma^-} \odot \boldsymbol{c}_{jl}^{\gamma^-} \tag{4.54}$$

$$\boldsymbol{o}_j^{\gamma} = \sigma\left(\boldsymbol{W}^{\gamma_\alpha^\omega(o)}\boldsymbol{x}_j + \sum_{l=1}^{2} \boldsymbol{R}_l^{\gamma_\alpha^\omega(o)}\boldsymbol{h}_{jl}^{-} + \boldsymbol{b}^{(o)}\right) \tag{4.55}$$

$$\boldsymbol{h}_j^{\gamma} = \boldsymbol{o}_j^{\gamma} \odot \tanh(\boldsymbol{c}_j^{\gamma}), \tag{4.56}$$

where $\boldsymbol{W}^{\gamma_\alpha^\omega(\cdot)}$ and $\boldsymbol{R}^{\gamma_\alpha^\omega(\cdot)}$ are the sliced coefficient network weight matrices.

4.4.0.1 Comparison of Number of Parameters Upon Slicing Operation

Here we investigate the effect of the slicing operation on the total number of parameters for the BBT-LSTM network. For the BBT-LSTM network with coefficient network weight matrices, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$, with depth α and a windowing factor of ω, the number of parameters with and without the slicing operation are; $\boldsymbol{W}^{\gamma_{\alpha}^{\omega}(\cdot)} : m\left(n - \lceil \frac{\omega}{2^{\alpha}} \rceil\right) = mn - m \lceil \frac{\omega}{2^{\alpha}} \rceil$, $\boldsymbol{R}^{\gamma_{\alpha}^{\omega}(\cdot)} : m\left(m - \lceil \frac{\omega}{2^{\alpha}} \rceil\right) = m^2 - m \lceil \frac{\omega}{2^{\alpha}} \rceil$ and $\boldsymbol{W}^{(\cdot)} : mn$, $\boldsymbol{R}^{(\cdot)} : mm$ respectively.

4.4.0.2 Range of $\lceil \frac{\omega}{2^{\alpha}} \rceil$

As defined and stated in the above subsection regarding the slicing operation and its effect on the number of parameters, the range of $\lceil \frac{\omega}{2^{\alpha}} \rceil$ can be stated as follows: For $\boldsymbol{W}^{\gamma_{\alpha}^{\omega}(\cdot)}$:

$$1 \leq \lceil \frac{\omega}{2^{\alpha}} \rceil < rank(\boldsymbol{W}^{\gamma_{\alpha}^{\omega}(\cdot)}) \tag{4.57}$$

or

$$1 \leq \lceil \frac{\omega}{2^{\alpha}} \rceil \leq n - 1. \tag{4.58}$$

Similarly, for $\boldsymbol{R}^{\gamma_{\alpha}^{\omega}(\cdot)}$:

$$1 \leq \lceil \frac{\omega}{2^{\alpha}} \rceil < rank(\boldsymbol{R}^{\gamma_{\alpha}^{\omega}(\cdot)}) \tag{4.59}$$

or

$$1 \leq \lceil \frac{\omega}{2^{\alpha}} \rceil \leq m - 1. \tag{4.60}$$

Proposition 4.3: *The total number of parameters for the BBT-LSTM networks with Slicing Operation are less than the total number of parameters of the BBT-LSTM networks without the Slicing Operation.*

Proof: Let $P_{(Sliced-BBT-LSTM)}$ and $P_{(BBT-LSTM)}$ be the total number of parameters of the BBT-LSTM networks with and without the Slicing Operation respectively. We have

$$\begin{aligned}
\frac{P_{(Sliced-BBT-LSTM)}}{P_{(BBT-LSTM)}} &= \frac{4(mn^{'} + mm^{'} + m)(2^{\alpha+1} - 1)}{4(mn + mm + m)(2^{\alpha+1} - 1)} \\
&\text{where } n' = n - \lceil \frac{\omega}{2^{\alpha}} \rceil \text{ and } m' = m - \lceil \frac{\omega}{2^{\alpha}} \rceil \\
&= \frac{mn^{'} + mm^{'} + m}{mn + mm + m} \\
&\approx \frac{mn - n\lceil \frac{\omega}{2^{\alpha}} \rceil + m^2 - m\lceil \frac{\omega}{2^{\alpha}} \rceil}{mn + mm} \\
&\approx \frac{m(n+m) - n\lceil \frac{\omega}{2^{\alpha}} \rceil - m\lceil \frac{\omega}{2^{\alpha}} \rceil}{m(m+n)} \\
&\approx \frac{m(n+m) - (m+n)\lceil \frac{\omega}{2^{\alpha}} \rceil}{m(m+n)} \\
&\approx 1 - \frac{\lceil \frac{\omega}{2^{\alpha}} \rceil}{m} \\
&\text{Using the range information from (4.60)} \\
&\leq 1 - \frac{m-1}{m} \\
&\leq \frac{1}{m} \qquad (4.61)
\end{aligned}$$

where (4.61) implies that $P_{(Sliced-BBT-LSTM)} \leq P_{(BBT-LSTM)}$.

□

4.4.1 WMF-Slicing Operation based BBT-LSTM Networks

In this subsection, we now apply the WMF to the Sliced-BBT-LSTM networks in order to further reduce the computational complexity and total number of parameters. We have two configurations for the WMF-Slicing operation based BBT-LSTM networks.

4.4.1.1 Configuration 1:

In this configuration, we first apply the WMF on the BBT-LSTM networks without slicing and then apply the slicing operation. For the BBT-LSTM network with coefficient weight matrices, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m \times m}$, applying the WMF yields $\boldsymbol{W}^{(\cdot)} \approx \boldsymbol{W}_1^{(\cdot)} \boldsymbol{W}_2^{(\cdot)}$ and $\boldsymbol{R}^{(\cdot)} \approx \boldsymbol{R}_1^{(\cdot)} \boldsymbol{R}_2^{(\cdot)}$ where $\boldsymbol{W}_1^{(\cdot)} \in \mathbb{R}^{m \times p}$, $\boldsymbol{W}_2^{(\cdot)} \in \mathbb{R}^{p \times n}$, $\boldsymbol{R}_1^{(\cdot)} \in \mathbb{R}^{m \times q}$ and $\boldsymbol{R}_2^{(\cdot)} \in \mathbb{R}^{q \times m}$.

Once we get the output of WMF, we then apply the slicing operation on the $\boldsymbol{W}_2^{(\cdot)}$ and $\boldsymbol{R}_2^{(\cdot)}$ yielding $\boldsymbol{W}_2^{\gamma_\alpha^\omega(\cdot)} \in \mathbb{R}^{p \times n^{'}}$ and $\boldsymbol{R}_2^{\gamma_\alpha^\omega(\cdot)} \in \mathbb{R}^{q \times m^{'}}$ respectively. Overall, we have the following number of parameters after applying WMF followed by slicing

$$\begin{aligned} \boldsymbol{W}^{\gamma_\alpha^\omega(\cdot)} \approx \boldsymbol{W}_1^{(\cdot)} \boldsymbol{W}_2^{\gamma_\alpha^\omega(\cdot)} &: mp + p\big(n - \lceil \frac{\omega}{2^\alpha} \rceil\big) \\ &: mp + pn - p \lceil \frac{\omega}{2^\alpha} \rceil \end{aligned}$$

$$\begin{aligned} \boldsymbol{R}^{\gamma_\alpha^\omega(\cdot)} \approx \boldsymbol{R}_1^{(\cdot)} \boldsymbol{R}_2^{\gamma_\alpha^\omega(\cdot)} &: mq + q\big(m - \lceil \frac{\omega}{2^\alpha} \rceil\big) \\ &: mq + qm - q \lceil \frac{\omega}{2^\alpha} \rceil \end{aligned}$$

Proposition 4.4: *The total number of parameters for the Sliced-WMF-BBT-LSTM networks (Configuration 1) are less than the total number of parameters of the WMF-BBT-LSTM networks.*

Proof: Let $P_{(Sliced-WMF-BBT-LSTM)}$ and $P_{(WMF-BBT-LSTM)}$ be the total number of parameters of the BBT-LSTM networks with and without the slicing operation respectively. We have

$$\begin{aligned}
\frac{P_{(Sliced-BBT-LSTM)}}{P_{(WMF-BBT-LSTM)}} &= \frac{4(mp+pn+2mq-(p+q)\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil+m)(2^{\alpha+1}-1)}{4(mp+pn+2mq+m)(2^{\alpha+1}-1)} \\
&= \frac{mp+pn+2mq-(p+q)\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil+m}{mp+pn+2mq+m} \\
&\approx 1-\frac{(p+q)\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil}{mp+pn+2mq} \\
&\text{let } m=n \\
&\approx 1-\frac{\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil}{2m} \\
&\leq 1-\frac{m-1}{2m} \\
&\leq \frac{1}{2}+\frac{1}{2m} \qquad (4.62)
\end{aligned}$$

where (4.62) implies that $P_{(Sliced-WMF-BBT-LSTM)} \leq P_{(WMF-BBT-LSTM)}$.

□

4.4.1.2 Configuration 2:

In this configuration, we apply the slicing operation to both of the sub-matrices achieved after applying the WMF. We achieve the following number of parameters

$$\begin{aligned}
\boldsymbol{W}^{\gamma^{\omega}_{\alpha}(\cdot)} \approx \boldsymbol{W}_1^{\gamma^{\omega}_{\alpha}(\cdot)}\boldsymbol{W}_2^{\gamma^{\omega}_{\alpha}(\cdot)} &: m\big(p-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big)+\big(p-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big)\big(n-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big) \\
&: mp+pn-m\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil-n\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil-p\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil+\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil^2
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{R}^{\gamma^{\omega}_{\alpha}(\cdot)} \approx \boldsymbol{R}_1^{\gamma^{\omega}_{\alpha}(\cdot)}\boldsymbol{R}_2^{\gamma^{\omega}_{\alpha}(\cdot)} &: m\big(q-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big)+\big(q-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big)\big(m-\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil\big) \\
&: 2mq-2m\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil-q\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil+\left\lceil\frac{\omega}{2^{\alpha}}\right\rceil^2
\end{aligned}$$

Proposition 4.5: *The total number of parameters for the Sliced-WMF-BBT-LSTM networks (Configuration 2) are less than the total number of parameters of the WMF-BBT-LSTM networks.*

Proof: Let $P_{(Sliced-WMF-BBT-LSTM)}$ and $P_{(WMF-BBT-LSTM)}$ be the total number of parameters of the BBT-LSTM networks (Configuration 2) with and without the slicing operation respectively. We have

$$\begin{aligned}
\frac{P_{(Sliced-BBT-LSTM)}}{P_{(WMF-BBT-LSTM)}} &= \frac{4(mp+pn+2mq-(p+q)\left\lceil\frac{\omega}{2^\alpha}\right\rceil-(3m+n)\left\lceil\frac{\omega}{2^\alpha}\right\rceil+m)(2^{\alpha+1}-1)}{4(mp+pn+2mq+m)(2^{\alpha+1}-1)} \\
&= \frac{mp+pn+2mq-(p+q)\left\lceil\frac{\omega}{2^\alpha}\right\rceil-(3m+n)\left\lceil\frac{\omega}{2^\alpha}\right\rceil+m}{mp+pn+2mq+m} \\
&\approx 1+\frac{2\left\lceil\frac{\omega}{2^\alpha}\right\rceil^2-(3m+n)\left\lceil\frac{\omega}{2^\alpha}\right\rceil-(p+q)\left\lceil\frac{\omega}{2^\alpha}\right\rceil}{mp+pn+2mq} \\
&\text{let } m=n \\
&\approx 1+\frac{2\left\lceil\frac{\omega}{2^\alpha}\right\rceil^2-4m\left\lceil\frac{\omega}{2^\alpha}\right\rceil-(p+q)\left\lceil\frac{\omega}{2^\alpha}\right\rceil}{2m(p+q)} \\
&\approx 1+\frac{\left\lceil\frac{\omega}{2^\alpha}\right\rceil^2}{m(p+q)}-\frac{\left\lceil\frac{\omega}{2^\alpha}\right\rceil}{p+q}-\frac{\left\lceil\frac{\omega}{2^\alpha}\right\rceil}{2m} \\
&\leq 1+\frac{(m-1)^2}{m(p+q)}-\frac{m-1}{p+q}-\frac{m-1}{2m} \\
&\leq \frac{1}{2}+\frac{1}{2m}-\frac{m-1}{p+q}+\frac{(m-1)^2}{m(p+q)} \\
&\text{if } m\text{is very large, then } m-1\approx m, \text{last two fractions cancel out yielding} \\
&\leq \frac{1}{2}+\frac{1}{2m} \qquad (4.63)
\end{aligned}$$

where (4.63) implies that $P_{(Sliced-WMF-BBT-LSTM)} \leq P_{(WMF-BBT-LSTM)}$.

□

4.5 Additive Gradient Updates for the BBT-LSTM Networks

In this section, we derive the additive gradient based updates for the BBT-LSTM networks. Before deriving the updates, we give some explanation about the terminologies used in the BBT-LSTM networks. Let us say that we have an α depth BBT-LSTM network where the number of final child nodes are $N = 2^{\alpha}$. At a depth of γ where $\gamma \in \{1, 2, \ldots, \alpha\}$, there are at a total of 2^{γ} child LSTM nodes. Out of the 2^{γ} child LSTM nodes, we base our gradient based calculations for the β child LSTM node where $\beta \in \{1, 2, \ldots, 2^{\gamma}\}$. Now at a depth of γ and at the β LSTM node, $\boldsymbol{f}_{\beta}^{\gamma}$, $\boldsymbol{f}_{\beta_1}^{\gamma-}$ and $\boldsymbol{f}_{\beta_2}^{\gamma-}$ represents the forget gate response for the β node at γ depth, left forget gate response coming to β node from $\gamma+1$ depth (left child node) and right forget gate response coming to β node from $\gamma+1$ depth (right child node) respectively. In our structure, we also have

$$\boldsymbol{f}_{\beta}^{\gamma} = \boldsymbol{f}_{\beta_1}^{(\gamma-1)-}.$$

Let us say that we have a regression problem at hand where we calculate the final estimate of the desired output as

$$\hat{d}_t = \boldsymbol{p}_t^T \boldsymbol{h}_t$$

where $\boldsymbol{p}_t \in \mathbb{R}^m$. Let us assume that we have a squared loss function, i.e., $\ell_t(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$, then the SGD update for the parameter θ is calculated as follows:

$$\theta_{t+1} = \theta_t + \mu \frac{\partial \ell_t}{\partial \theta} = \theta_t - 2\mu(d_t - \hat{d}_t)\boldsymbol{p}_t^T \frac{\partial \boldsymbol{h}_t}{\partial \theta}, \tag{4.64}$$

where μ is the learning rate parameter.

Let $\theta = w_{ij}^{(i)}(\gamma, \beta)$, where $w_{ij}^{(i)}(\gamma, \beta)$ is the i^{th} row and j^{th} column entry of the input gate matrix corresponding to the β node in the γ depth, then $\dfrac{\partial \boldsymbol{h}_t}{\partial w_{ij}^{(i)}(\gamma, \beta)}$ is calculated as follows:

$$\frac{\partial \boldsymbol{h}_t}{\partial w_{ij}^{(i)}(\gamma,\beta)} = \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_t^{+,-}} \frac{\partial \boldsymbol{h}_t^{+,-}}{\partial \boldsymbol{h}_{\{1,2\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,\ldots,4\}}^{+,-}} \cdots \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,k-1\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,k+1\}}^{+,-}} \cdots \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma-1}\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma}\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma}\}}^{+,-}}{\partial w_{ij}^{(i)}(\gamma,\beta)}, \tag{4.65}$$

where $\boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}$ is the corresponding state vector response in the path to the β node in the γ depth. In $\boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}$, the superscript $+$ and $-$ denotes the corresponding left and right path to the child nodes.

Specifically, for the β node, the expression $\dfrac{\partial \boldsymbol{h}_\beta}{\partial w_{ij}^{(i)}(\gamma,\beta)}$ can be calculated as follows:

$$\frac{\partial \boldsymbol{h}_\beta}{\partial w_{ij}^{(i)}(\gamma,\beta)} = \boldsymbol{\Pi}_\beta^{(\boldsymbol{o})} \boldsymbol{\Pi}_\beta^{(\tanh'(\boldsymbol{c}_\beta))} \frac{\partial \boldsymbol{c}_\beta}{\partial w_{ij}^{(i)}(\gamma,\beta)} \tag{4.66}$$

where $\dfrac{\partial \boldsymbol{c}_\beta}{\partial w_{ij}^{(i)}(\gamma,\beta)}$ is calculated as follows:

$$\frac{\partial \boldsymbol{c}_\beta}{\partial w_{ij}^{(i)}(\gamma,\beta)} = \boldsymbol{\Pi}_\beta^{(\sigma'(\boldsymbol{\rho}))} x_j \tilde{\boldsymbol{c}}_\beta \tag{4.67}$$

where $\boldsymbol{\rho} = \boldsymbol{W}^{(i)} \boldsymbol{x}_\beta + \sum_{l=1}^{2} \boldsymbol{R}_l^{(i)} \boldsymbol{h}_{\beta l}^{-} + \boldsymbol{b}^{(i)}$.

Similarly, for $\theta = r_{1,ij}^{(i)}(\gamma,\beta)$, where $r_{1,ij}^{(i)}(\gamma,\beta)$ is the i^{th} row and j^{th} column entry of the recurrent input gate matrix corresponding to the β node in the γ depth, then $\dfrac{\partial \boldsymbol{h}_t}{\partial r_{1,ij}^{(i)}(\gamma,\beta)}$ is calculated as follows:

$$\frac{\partial \boldsymbol{h}_t}{\partial r_{1,ij}^{(i)}(\gamma,\beta)} = \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_t^{+,-}} \frac{\partial \boldsymbol{h}_t^{+,-}}{\partial \boldsymbol{h}_{\{1,2\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,\ldots,4\}}^{+,-}} \cdots \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,k-1\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,k\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,k+1\}}^{+,-}} \cdots \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma-1}\}}^{+,-}}{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma}\}}^{+,-}} \frac{\partial \boldsymbol{h}_{\{1,2,\ldots,2^{\gamma}\}}^{+,-}}{\partial r_{1,ij}^{(i)}(\gamma,\beta)}, \tag{4.68}$$

where specifically $\frac{\partial \boldsymbol{h}_\beta}{\partial r_{1,ij}^{(i)}(\gamma,\beta)}$ is calculated as follows:

$$\frac{\partial \boldsymbol{h}_\beta}{\partial r_{1,ij}^{(i)}(\gamma,\beta)} = \boldsymbol{\Pi}_\beta^{(\boldsymbol{o})} \boldsymbol{\Pi}_\beta^{(\tanh'(\boldsymbol{c}_\beta))} \frac{\partial \boldsymbol{c}_\beta}{\partial r_{1,ij}^{(i)}(\gamma,\beta)} \tag{4.69}$$

where $\frac{\partial \boldsymbol{c}_\beta}{\partial r_{1,ij}^{(i)}(\gamma,\beta)}$ is calculated as follows:

$$\frac{\partial \boldsymbol{c}_\beta}{\partial r_{1,ij}^{(i)}(\gamma,\beta)} = \boldsymbol{\Pi}_\beta^{(\sigma'(\boldsymbol{\rho}))} h_{\beta_1,j}^{-} \tilde{\boldsymbol{c}}_\beta. \tag{4.70}$$

Table 4.2: Comparison of Computational Complexity for all the Algorithms. Here, $p << \min(m,n)$ and $q << m$.

Algorithms	**Parameters**
LSTM	$4(mn+mm+m)$
N-ary Tree LSTM	$4(mn+mm+m)(N+1)$
BBT-LSTM	$4(mn+mm+m)(2^{\alpha+1}-1)$
ef-BBT-LSTM	$4(mn+mm+3m)(2^{\alpha+1}-1)$
WMF-BBT-LSTM	$4(mp+pn+2mq+m)(2^{\alpha+1}-1)$
ef-WMF-BBT-LSTM	$4(mp+pn+2mq+3m)(2^{\alpha+1}-1)$
Sliced-BBT	$4(mn-n\lceil\frac{\omega}{2^\alpha}\rceil+m^2-m\lceil\frac{\omega}{2^\alpha}\rceil)(2^{\alpha+1}-1)$
Sliced-BBT-WMF-LSTM-1	$4(mp+pn+2mq-(p+q)\lceil\frac{\omega}{2^\alpha}\rceil+m)(2^{\alpha+1}-1)$
Sliced-BBT-WMF-LSTM-2	$4(mp+pn+2mq-(p+q)\lceil\frac{\omega}{2^\alpha}\rceil-(3m+n)\lceil\frac{\omega}{2^\alpha}\rceil+m)(2^{\alpha+1}-1)$

4.6 Experiments

In this section, we demonstrate the performance of the proposed regression algorithm, i.e., BBT-LSTM with simple LSTM network and $N-$ary Tree LSTM networks. We validate the performance using a real-life financial data set and kinematics data set. We use the LSTM networks with only one hidden layer in all the models. For the parameters update, we use stochastic gradient descent (SGD) algorithm. In order to provide a fair experimental environment, we use $\eta = 0.1$ as our learning rate.

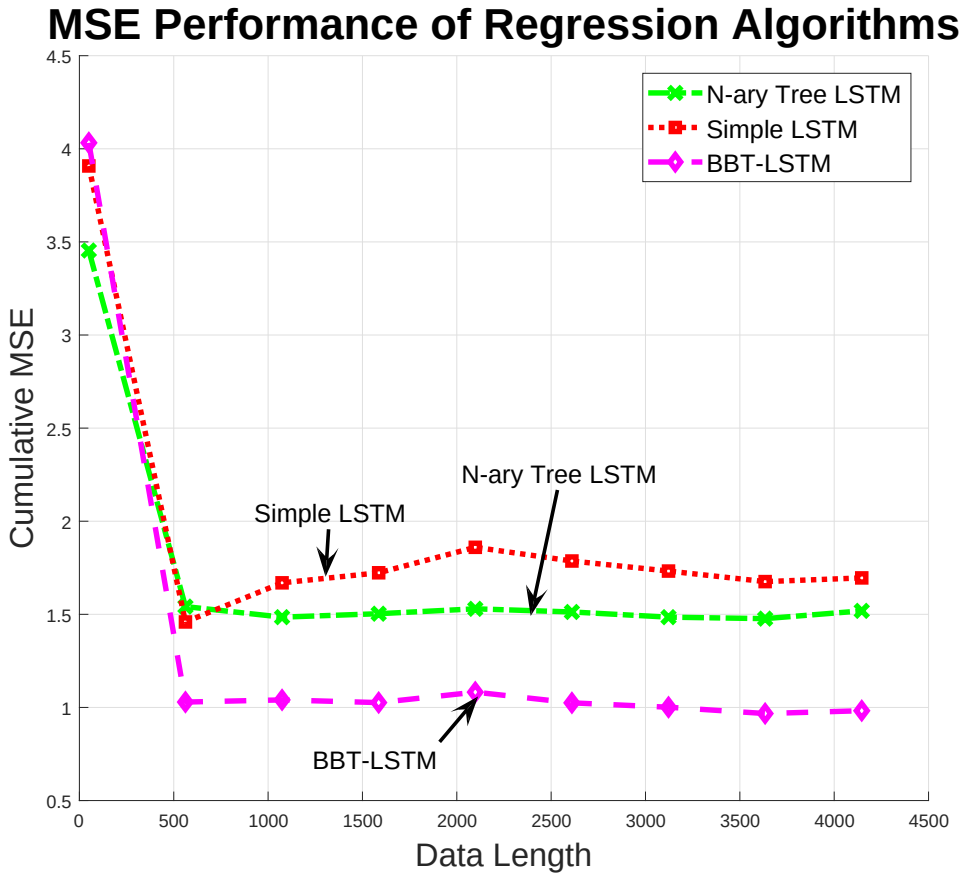

Figure 4.4: MSE performance comparison of proposed BBT-LSTM network with Simple LSTM and N-ary Tree LSTM networks using the Alcoa Corp. stock price data set.

For the financial data set, we use Alcoa corporation stock price data set [37]. In this data set, we aim to predict the future stock price based on past values.

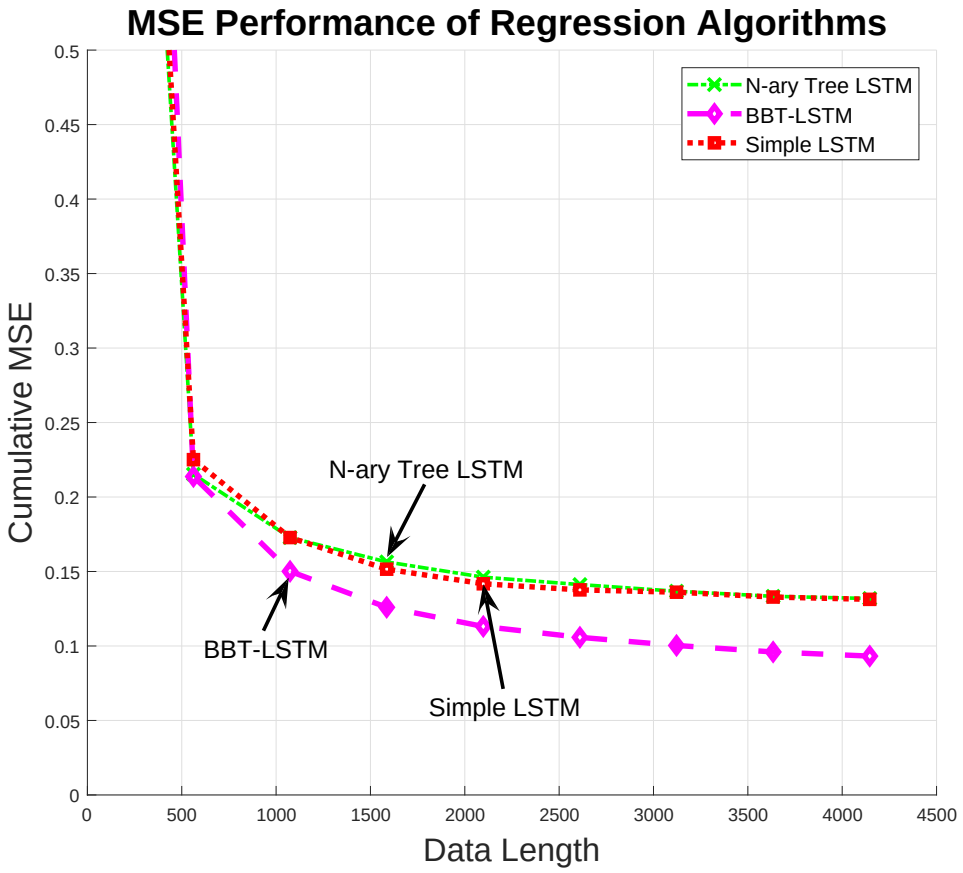

Figure 4.5: MSE performance comparison of proposed BBT-LSTM network with Simple LSTM and N-ary Tree LSTM networks using the Kinematics data set.

In order to predict the one-step-ahead stock price value, we use past 5 days stock values. We use $N = 4$ for Tree LSTM structure. The input vector to the simple LSTM and Tree LSTM network is $\boldsymbol{x}_t \in \mathbb{R}^5$. For the proposed BBT-LSTM framework, we use depth $\alpha = 2$ and window factor $w = 1$. We have binary splitting at each LSTM parent/child node. We evaluate the regression performance by using mean square error (MSE) as our performance criterion. In Fig. 4.4, the accumulated MSE for the regression performance is shown for all the regression algorithms. We observe from the Fig. 4.4 that the proposed BBT-LSTM network for $\alpha = 2$ and $w = 1$ performs exceptionally well as compared to the other algorithms. Tree LSTM performed better than simple LSTM network. In Table 4.3, we list down the steady-state error for all the regression algorithms. Table 4.4 shows the timing profiles for all the algorithms. We observe from

Table 4.4 that BBT-LSTM takes a little bit more time than Tree LSTM and simple LSTM networks. While on the other hand, BBT-LSTM has the minimum steady-state error as shown in Table 4.3.

We also experiment with Kinematics data set where we have the simulated data of the eight-link robotic arm [22]. In this case, we intend to predict the distance of the disturbing source from the target. We have an input vector of $\boldsymbol{x}_t \in \mathbb{R}^8$. We use $N = 8$ as the number of child nodes for the Tree LSTM network. For the BBT-LSTM network, we use depth $\alpha = 2$ and windowing factor $w = 2$.

From Fig. 4.5, we observe that again the BBT-LSTM network shows the best overall performance. While Tree LSTM and simple LSTM network show almost similar performance. The steady-state error and timing profile for kinematic data set is shown in Table 4.3 and Table 4.4 respectively.

Table 4.3: Steady State Error performance for all the regression algorithms for Alcoa, Kinematics, Protein Tertiary and Puma8NH data sets

Algorithms / Data Sets	**Alcoa (N=4) rank = 2** $\lceil(\cdot)\rceil = 1$	**Kinematics (N=8) rank = 3** $\lceil(\cdot)\rceil = 2$	**Protein (N=8) rank =3** $\lceil(\cdot)\rceil = 2$	**Puma (N=8) rank = 3** $\lceil(\cdot)\rceil = 1$
LSTM	1.6960	0.1313	0.2569	0.1284
N-ary Tree-LSTM	1.5190	0.1319	0.2318	0.1058
BBT-LSTM	0.9820	0.0930	0.1987	0.0895
ef-BBT-LSTM	0.9825	0.0996	0.2010	0.1000
WMF-BBT-LSTM	0.9823	0.1017	0.1999	0.1011
ef-WMF-BBT-LSTM	0.9824	0.1008	0.2004	0.1008
Sliced-BBT	0.9823	0.1083	0.1997	0.9991
Sliced-BBT-WMF-LSTM-1	0.9824	0.1090	0.2011	0.1040
Sliced-BBT-WMF-LSTM-2	0.9827	0.1094	0.2106	0.1051

We now compare the computational complexity of the algorithms. For this purpose, we give the total number of parameters for the Algorithms. In Table

Table 4.4: Timing Profile (in seconds) for all the regression algorithms for Alcoa, Kinematics, Protein Tertiary and Puma8NH data sets

Algorithms / Data Sets	**Alcoa (N=4) rank = 2** $\lceil(\cdot)\rceil = 1$	**Kinematics (N=8) rank = 3** $\lceil(\cdot)\rceil = 2$	**Protein (N=8) rank =3** $\lceil(\cdot)\rceil = 2$	**Puma (N=8) rank = 3** $\lceil(\cdot)\rceil = 1$
LSTM	26.1230	45.2627	430.0817	450.1800
N-ary Tree-LSTM	80.1874	151.1211	2050.4848	2160.8795
BBT-LSTM	120.8266	226.6817	2987.0004	3200.1474
ef-BBT-LSTM	140.6789	250.7800	3248.0981	3501.1111
WMF-BBT-LSTM	98.6741	170.1576	2313.3314	2699.2142
ef-WMF-BBT-LSTM	125.1428	199.1777	2643.1999	2889.9815
Sliced-BBT	76.1578	164.1287	2200.0214	2465.6541
Sliced-BBT-WMF-LSTM-1	60.2468	151.0006	2156.6544	2301.1119
Sliced-BBT-WMF-LSTM-2	37.1697	99.7812	1141.1745	1754.9512

4.5, we list down the total number of parameters for all the four data sets. From Table 4.5, it is obvious that the Sliced-BBT-LSTM networks have the lowest number of parameters and still having closest to the best possible performance as shown in Table 4.3 and having the optimum training time (in seconds) as shown in Table 4.4.

Remark 4.5: *We now perform the similar set of experiments (as conducted above using the LSTM networks) using the GRU networks and list down the performance. We investigate the steady state error performance, training time profile(in seconds) and the total number of parameters in Appendix C*

Table 4.5: Total number of parameters for all the regression algorithms for Alcoa, Kinematics, Protein Tertiary and Puma8NH data sets

Algorithms / Data Sets	**Alcoa (N=4) rank = 2** $\lceil(\cdot)\rceil = 1$	**Kinematics (N=8) rank = 3** $\lceil(\cdot)\rceil = 2$	**Protein (N=8) rank =3** $\lceil(\cdot)\rceil = 2$	**Puma (N=8) rank = 3** $\lceil(\cdot)\rceil = 1$
LSTM	220	544	684	684
N-ary Tree-LSTM	1100	4896	6156	6156
BBT-LSTM	1540	8160	10260	10260
ef-BBT-LSTM	1820	9120	11340	11340
WMF-BBT-LSTM	700	3360	3780	3780
ef-WMF-BBT-LSTM	980	4320	4860	4860
Sliced-BBT	1260	6240	7020	8100
Sliced-BBT-WMF-LSTM-1	1148	5520	6300	6660
Sliced-BBT-WMF-LSTM-2	644	2280	2460	4620

Chapter 5

Minimal Hop-Long Short Term Memory (MH-LSTM) Networks

In this chapter, we introduce a low complexity version of the LSTM networks that operates by employing a minimum number of hops over the input data sequence. We call these low complexity networks as Minimal Hop Long Short Term Memory (MH-LSTM) networks. The MH-LSTM networks have low time and computational complexity. The main idea is to reduce the total number of hops over the input data sequence significantly but still achieving nearest to the best possible performance. For example, let us say we have an input data sequence of length 100, then instead of hopping over each time instance, i.e., 100 times, we hop let us say 10 time which is ten times less than the original one and still achieving closest to the best overall performance.

5.1 Minimal Hop-Recurrent Neural Network (MH-RNN)

We first develop the minimal hop mathematical framework for the RNNs and then extend this framework for the LSTM networks in a similar fashion. Given

that we have a incoming data sequence $\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}, \ldots, \boldsymbol{x}_{t_{\zeta_t}}]$, where $\boldsymbol{x}_{t_i} \in \mathbb{R}^p, \forall i \in \{1, 2, \ldots, \zeta_t\}, \boldsymbol{X}_t \in \mathbb{R}^{p \times \zeta_t}$ and $\zeta_t \in \mathbb{Z}^+$ represents the length of the input data sequence which may be variable (ζ_t) or fixed ($\zeta_t = T$).

When we pass the input data sequence $\boldsymbol{X}_t$ through the RNN, the internal equations for the i^{th} block of the RNN is as follows [14]:

$$\boldsymbol{h}_{t_i} = \sigma(\boldsymbol{W}^{(h)}\boldsymbol{x}_{t_i} + \boldsymbol{R}^{(h)}\boldsymbol{h}_{t-1_i}) \tag{5.1}$$

$$\boldsymbol{y}_{t_i} = \sigma(\boldsymbol{W}^{(y)}\boldsymbol{h}_{t_i}) \tag{5.2}$$

where $\boldsymbol{x}_{t_i} \in \mathbb{R}^p$, $\boldsymbol{h}_{t_i} \in \mathbb{R}^m$ and $\boldsymbol{y}_{t_i} \in \mathbb{R}^m$ are the input, state and output vector for the i^{th} block of the RNN.

Now for a total of ζ_t regular time steps (hops), we divide down the number of hops to α where $\alpha << \zeta_t$ with a hopping distance of Δ such that

$$\frac{\zeta_t}{\Delta} = \alpha \tag{5.3}$$

$$\zeta_t = \alpha\Delta \tag{5.4}$$

where $\Delta, \alpha, \zeta_t \in \mathbb{Z}^+$. This Δ hopping over the input data sequence can be seen in Fig. . From Fig. , the RNN equations for the $i\Delta$ time hopping time instance are as follows:

$$\boldsymbol{h}_{t_{i\Delta}} = \sigma(\boldsymbol{W}^{(h)}\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{R}^{(h)}\boldsymbol{h}_{t-1_{i\Delta}}) \tag{5.5}$$

$$\boldsymbol{h}_t = \kappa(\{\boldsymbol{h}_{t_1}, \boldsymbol{h}_{t_{i\Delta}}\}_{i=1}^{\alpha}) \tag{5.6}$$

where $\boldsymbol{x}_{t_{i\Delta}}$ and $\boldsymbol{h}_{t_{i\Delta}}$ are the input and state vectors for the $i\Delta$ block of the RNN and $\boldsymbol{h}_t = \kappa(\cdot)$ where $\kappa(\cdot)$ represents the mean, max and last pooling operation.

5.2 Minimal Hop-Long Short Term Memory (MH-LSTM) Networks

In this section, we will extend the minimal hopping mathematical framework on the LSTM networks similar to MH-RNN. For the input data sequence $\boldsymbol{X}_t$ of

length ζ_t with a hopping distance of Δ and number of down sized hops α, the internal LSTM equations for the $i\Delta$ time instance are as follows:

$$\tilde{\boldsymbol{c}}_{t_{i\Delta}} = \tanh\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1_{i\Delta}} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{5.7}$$

$$\boldsymbol{i}_{t_{i\Delta}} = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1_{i\Delta}} + \boldsymbol{b}^{(i)}\right) \tag{5.8}$$

$$\boldsymbol{f}_{t_{i\Delta}} = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1_{i\Delta}} + \boldsymbol{b}^{(f)}\right) \tag{5.9}$$

$$\boldsymbol{c}_{t_{i\Delta}} = \boldsymbol{i}_{t_{i\Delta}} \odot \tilde{\boldsymbol{c}}_{t_{i\Delta}} + \boldsymbol{f}_{t_{i\Delta}} \odot \boldsymbol{c}_{t-1_{i\Delta}} \tag{5.10}$$

$$\boldsymbol{o}_{t_{i\Delta}} = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1_{i\Delta}} + \boldsymbol{b}^{(o)}\right) \tag{5.11}$$

$$\boldsymbol{h}_{t_{i\Delta}} = \boldsymbol{o}_{t_{i\Delta}} \odot \tanh(\boldsymbol{c}_{t_{i\Delta}}), \tag{5.12}$$

where $\boldsymbol{c}_{t_{i\Delta}} \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_{t_{i\Delta}} \in \mathbb{R}^p$ is the input vector and $\boldsymbol{h}_{t_{i\Delta}} \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_{t_{i\Delta}}$, $\boldsymbol{f}_{t_{i\Delta}}$ and $\boldsymbol{o}_{t_{i\Delta}}$ are the input, forget and output gates, respectively. The sigmoid function $\sigma(\cdot)$ and $\tanh(\cdot)$ applies pointwise to the vector elements. The weight matrices and bias vectors have the following dimensions, i.e., $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{m\times p}$, $\boldsymbol{R}^{(\cdot)} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{b}^{(\cdot)} \in \mathbb{R}^m$.

Remark 5.1: *From Fig.* , $\boldsymbol{x}_{t_{(i+1)\Delta}}$ can be of the following forms, i.e.,
(1) mean pooled version : $\dfrac{\boldsymbol{x}_{t_{i\Delta}} + \boldsymbol{x}_{t_{i\Delta}+1} + \ldots + \boldsymbol{x}_{t_{(i+1)\Delta}}}{\Delta + 1}$,
(2) max pooled version : $\max(\boldsymbol{x}_{t_{(i)\Delta}}, \boldsymbol{x}_{t_{i\Delta}+1}, \ldots, \boldsymbol{x}_{t_{(i+1)\Delta}})$ and
(3) last pooled version : $\boldsymbol{x}_{t_{(i+1)\Delta}}$.

5.3 Selecting the Value of Hopping Distance Δ

In this section, we devise some ways in order to select the appropriate value of hopping distance Δ in order to achieve best overall performance. We work on two criterion for selecting the value of Δ namely fixed and variable Δ selection methods. The details of these methods are as follows:

5.3.1 Fixed Δ Method

In this method, as obvious from the name we fix the value of the Δ and carry out the experiment. For this method, in order to achieve the best possible performance, we cross validate the experiment using different values of Δ. This method is quite trivial and trial and error based method. This method is not much reliable as it may require a lot of time and computational complexity in order to select the best value for the Δ.

5.3.2 Variable Δ Method

In this method, during an experiment, we use varying (both increasing or decreasing) value of Δ. The variable Δ value is continuously adjusted during the experimental procedure for achieving the best result.

We further aid the selection of variable Δ value by introducing the Similarity measure. We define a Similarity Matrix (SM) that contains the similarities of the data instances. For example, in terms of tweets or movie review based data set, we have the similarities of the words among themselves. Based on the similarity values, we either increase or decrease the hopping distance Δ value.

5.3.2.1 Similarity Matrix (SM) and its Computation

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ and $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ where $\boldsymbol{x}_i \in \mathbb{R}^m$ and $\boldsymbol{y}_i \in \mathbb{R}^m$, then the SM can be written as follows [38]:

$$\begin{bmatrix} s_{x_1y_1} & s_{x_1y_2} & s_{x_1y_3} & s_{x_1y_4} & \cdots & s_{x_1y_n} \\ & s_{x_2y_2} & s_{x_2y_3} & s_{x_2y_4} & \cdots & s_{x_2y_n} \\ & & s_{x_3y_3} & s_{x_3y_4} & \cdots & s_{x_3y_n} \\ & & & \ddots & \cdots & \vdots \\ & & & & \ddots & s_{x_{n-1}y_n} \\ & & & & & s_{x_ny_n} \end{bmatrix}$$

where $s_{x_i y_j} \forall i, j \in \{1, 2, \ldots, n\}$ is the similarity value of $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$. The SM is a triangular matrix (either lower or upper triangular matrix). The total number of calculations required for its computation is $\frac{n(n-1)}{2}$.

In order to calculate the similarity matrix entries specifically for $s_{x_i y_j}$, we use various information retrieval metrics like dice coefficient [39], cosine coefficient [40], jaccard coefficient [41] and inner product [42] as defined follows:

- Dice Coefficient : $\frac{2 \sum_{\beta=1}^{m} x_\beta y_\beta}{\sum_{\beta=1}^{m} x_\beta^2 + \sum_{\beta=1}^{m} y_\beta^2}$
- Cosine Coefficient : $\frac{\sum_{\beta=1}^{m} x_\beta y_\beta}{\sqrt{\sum_{\beta=1}^{m} x_\beta^2 * \sum_{\beta=1}^{m} y_\beta^2}}$
- Jaccard Coefficient : : $\frac{\sum_{\beta=1}^{m} x_\beta y_\beta}{\sum_{\beta=1}^{m} x_\beta^2 + \sum_{\beta=1}^{m} y_\beta^2 - \sum_{\beta=1}^{m} x_\beta y_\beta}$
- Inner Product : $\sum_{\beta=1}^{m} x_\beta y_\beta$.

Remark 5.2: *Since the total number of vocabulary or distinct words / characters for a particular data set are too large, so we cannot create the overall SM in an offline manner because of the memory constraints. The remedy to this problem is to create the sub-similarity matrices (sub-SMs) at each online processing of the input data sequence. This sub-SM is constituted by calculating the similarities for the distinct words for the current input data sequence currently under process. For example, for the tweets, for the worst case, the maximum size of the SM will be* 140×140*, but we need only* 139×70 *computations at max because of the triangular nature of the SM.*

5.3.2.2 Hopping Distance Value Selection Criterion

We now shed some light on how to select the Δ values based on the similarity score obtained from the SM using various metrics. We start with some initially defined Δ value at the start of the input sequence $\boldsymbol{X}_t$ let us say for example we

have a tweets data set. We give an initial Δ values at the starting word of the tweet and then we aim to hop by Δ by utilizing the similarity information. For this purpose, we compare the similarity score of the first word of the tweet with the subsequent words and hop at the most similar word, i.e., whose similarity score is alike by some threshold. We define this threshold as similarity threshold (δ) defined as $\delta = \gamma\epsilon$ where ϵ is the similarity difference value assigned usually as ± 0.01 or ± 0.001 or some other user defined value. This ϵ value is dependent on the experimental setup, parameters like different similarity metrics and on various kinds and nature of data sets. Whereas, $\gamma \in \mathbb{Z}^+$ is the scaling constant.

5.4 Time and Computational Complexity

In this section, we compare the computational complexity of the MH-LSTM networks with the simple LSRTm networks. Let us say for a sequence (for example a tweet) $\boldsymbol{X}_t$ of fixed length $\zeta_t = T$. Then the total number of computations required for parsing one complete sequence of length T using the simple LSTM networks is

$$4(mn + mm + m) * T \tag{5.13}$$

where $4(mn+mm+m)$ is the number of computations for one time step instance of the LSTM networks.

Similarly, for the MH-LSTM networks with hopping distance Δ, number of hops α and the fixed length of the input sequence $\zeta_t = T$, the total number of computations is as follows:

$$\begin{aligned} &4(mn + mm + m) * \alpha \\ &4(mn + mm + m) * \frac{T}{\Delta} \end{aligned} \tag{5.14}$$

where $\alpha = \dfrac{T}{\Delta}$.

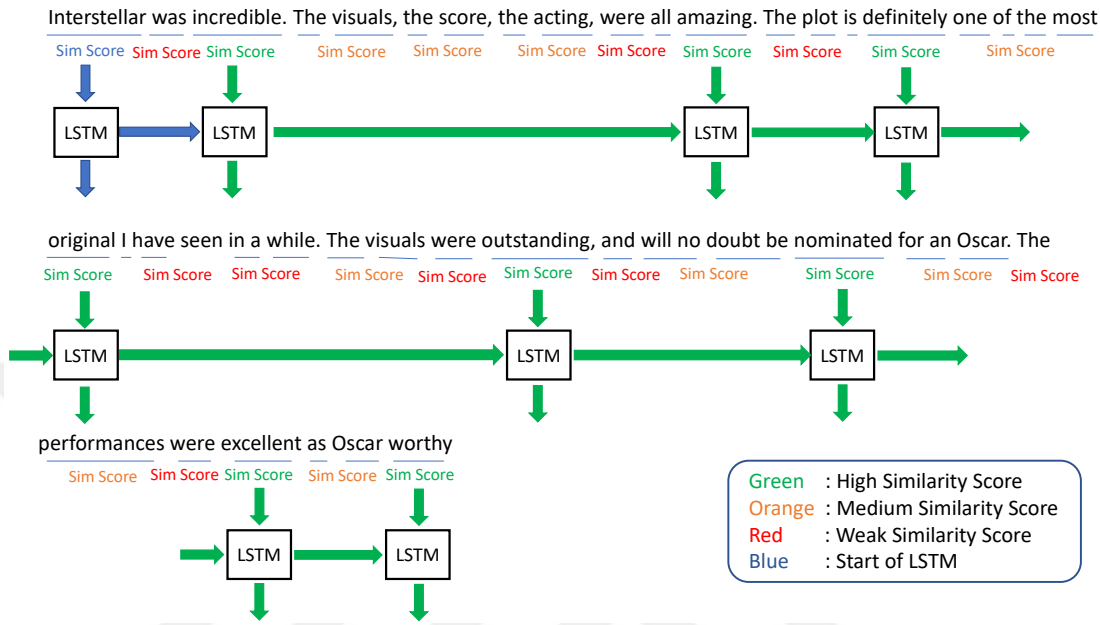

Figure 5.1: IMDB movie review sample for Interstellar given by a random user. Movie review length = 47 and based on Similarity Scores, only 9 LSTM blocks are used instead of 47.

Remark 5.3: *By comparing (5.14) and (5.13), we observe that the computational complexity of the MH-LSTM networks is Δ times less than that of the simple LSTM networks. In the experiment section, we observe that the MH-LSTM networks achieves the performance closest to the best possible performance shown by the simple LSTM networks. In terms of time complexity, it is quite evident from (5.14) and (5.13) that the time complexity of the MH-LSTM networks is less than the simple LSTM networks.*

5.5 Experiments

In this section, we validate the performance of the proposed algorithms using the three real-life data sets namely: Twitter US Airline Sentiment Data Set [43], Yelp Open Round-10 Data Set [44] and IMDB Movie Reviews Data Set [45]. The basic Python libraries used in the experiments are as follows

Figure 5.2: 2D Visualization of the positive and negative sentiment words depicted by two 2 clusters. The cluster with blue words shows the positive sentiments while the cluster with red words shows the negative sentiments.

- NumPy
- Pandas
- Keras
- NLTK
- Scikit-learn

5.5.1 Twitter US Airline Sentiment Data Set

First, we experiment on Twitter US Airline Sentiment Data Set where we have the tweets from the people regarding their comments on the US airline services on February 15. The users were asked to tweet their concerns about the US airline companies in order to get a feedback. This feedback was collected in terms of positive, negative and neutral terms. Here, we use the version where we have only the positive and negative tweets. Our main aim is to perform sentiment analysis using simple LSTM networks along with MH-LSTM networks and compare their overall performance, time and computational complexity. In the data set, we have a total of 14641 tweets out of which 9178 are negative, 2363 are positive and 3100 are neutral. For binary sentiment analysis, we discard the 3100 neutral tweets. So now we have a total of 11541 tweets. Based on the tweets, we first give an overview statistics of the best and worst airlines based on the number of tweets as follows:

Table 5.1: Statistics of US Airlines Based on Tweets

Worst Airline	**Number of Tweets**		**Best Airline**	**Number of Tweets**
United	2633		**Southwest**	570
US Airways	2263		**Delta**	544
American	1960		**United**	492
Southwest	1186		**American**	336
Delta	955		**US Airways**	269
Virgin America	181		**Virgin America**	152

As a sample of the data set, we first show the first 10 tweets form the original data set in Fig. 5.3.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone	
2	5.70306E+17	neutral	1			Virgin America		cairdin		0	@VirginAmerica What @c		2/24/2015 11:35		Eastern Time (US & Canada)	
3	5.70301E+17	positive	0.3486		0	Virgin America		jnardino		0	@VirginAmerica plus you'		2/24/2015 11:15		Pacific Time (US & Canada)	
4	5.70301E+17	neutral	0.6837			Virgin America		yvonnalynn		0	@VirginAmerica I didn't t		2/24/2015 11:15	Lets Play	Central Time (US & Canada)	
5	5.70301E+17	negative	1	Bad Flight	0.7033	Virgin America		jnardino		0	@VirginAmerica it's really		2/24/2015 11:15		Pacific Time (US & Canada)	
6	5.70301E+17	negative	1	Can't Tell	1	Virgin America		jnardino		0	@VirginAmerica and it's a		2/24/2015 11:14		Pacific Time (US & Canada)	
7	5.70301E+17	negative	1	Can't Tell	0.6842	Virgin America		jnardino		0	@VirginA		2/24/2015 11:14		Pacific Time (US & Canada)	
8	5.70301E+17	positive	0.6745		0	Virgin America		cjmcginnis		0	@VirginAmerica yes, near		2/24/2015 11:13	San Francisco CA	Pacific Time (US & Canada)	
9	5.703E+17	neutral	0.634			Virgin America		pilot		0	@VirginAmerica Really m		2/24/2015 11:12	Los Angeles	Pacific Time (US & Canada)	
10	5.703E+17	positive	0.6559			Virgin America		dhepburn		0	@virginamerica Well, I di		2/24/2015 11:11	San Diego	Pacific Time (US & Canada)	

Figure 5.3: Sample Tweet of the Airline US Twitter Data Set

In the experimental setup, we first use the simple LSTM networks followed by a

dense layer with softmax activation in order to execute binary sentiment analysis. For the embedding matrix, we experiment with different dimensions like $64, 128$ and 256 and record the maximum accuracy achieved. The maximum input length is 32. We add a dropout layer with a factor of 0.5. We add a single layer of the LSTM networks followed by a dense layer. For the compiling, we chose the loss as binary crossentropy and *adam* as the optimizer. We select a batch size of 512 and introduce early stopping while fitting the model. For the early stopping, we choose the stopping criteria as the validation loss where if the validation loss does not increase or decrease by 0.001 for 5 epochs then the training stops. In the end, we then test the fitted model and mention the positive and negative accuracy. After the training is done, we save the embedding matrix separately which will be used in calculating the similarity values in the MH-LSTM networks.

Next we experiment with the MH-LSTM networks. We make use of both the fixed and variable Δ hopping distance. Keeping the parameters same as in the previous simple LSTM networks for a fair experimental setup, we repeat the binary sentiment analysis with different values of Δ in the MH-LSTM networks. The maximum input length of the tweets in the data set is 32. We experiment with different values of Δ like $\Delta = \{2, 4, 8\}$ for the Fixed Δ-MH-LSTM networks (FΔ-MH-LSTM). Similarly, for the Variable Δ-MH-LSTM networks (VΔ-MH-LSTM), we switch between different Δ values in the range $[2, 8]$. The criteria for using different variable Δ value is the difference in training, i.e., ΔE. If the ΔE keeps decreasing for α samples, then we increase the Δ value by one. If upon increasing the Δ value too much, the error instead of staying stable or decreasing (very unlikely) or very slightly increasing (likely) increases by β threshold, then we decrease the Δ value and continue decreasing until we have a decreasing error trend.

We now take into account he effect of similarity values on the Δ hopping distance and call these networks as Similarity Δ-MH-LSTM networks, i.e., SΔ-MH-LSTM networks. From the embedding matrix saved in the simple LSTM networks experiments, we calculate several coefficients like dice, cosine, jacord and inner product. Intuitively, similar positive sentiment words will have a higher positive coefficient values and vice versa. This can be easily seen in Fig. , where

Table 5.2: Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple LSTM networks and MH-LSTM networks. For the FΔ-MH-LSTM networks, we have accuracies and AUC scores for $\Delta = \{2, 4, 8\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN	78.12	91.67	0.8813
LSTM	89.29	96.01	0.9687
FΔ-MH-LSTM	83.15 82.11 77.48	93.72 91.12 89.88	0.9371 0.9115 0.8641
RΔ-MH-LSTM	84.99	92.63	0.9398
SΔ-MH-LSTM (Dice)	87.15	94.11	0.9499
SΔ-MH-LSTM (Cosine)	89.01	94.87	0.9555
SΔ-MH-LSTM (Jacord)	87.65	93.70	0.9500
SΔ-MH-LSTM (Inner)	78.18	81.20	0.8650

we have 2 big clusters signifying the positive and negative words. In a particular, we first create a Similarity Matrix (in this experiment 32×32 but we only need to have 31×16 entries because of the triangular property of the SM). After the generation of SM, we then compare the similarity scores and hop to that index of the sequence that are more similar, i.e., either positive or negative, as shown in Fig. .

The accuracies and the AUC-scores for all the algorithms is shown in Table 5.2.

5.5.2 Yelp Round-10 Review Data Set

In this subsection, we repeat the above experimental setup on Yelp Round-10 Reviews data set. The main aim is to classify the reviews into positive and negative reviews. The maximum length of the reviews is 50. The reviews shorter than 50 are padded with zeros in order to make the length of the sequence to be

same.

Table 5.3: Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple LSTM networks and MH-LSTM networks. For the FΔ-MH-LSTM networks, we have accuracies and AUC scores for $\Delta = \{2, 4, 8, 10\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN	91.03	85.17	0.8512
LSTM	97.99	94.13	0.9788
FΔ-MH-LSTM	93.11	90.76	0.9412
	92.06	88.97	0.9400
	90.05	86.31	0.9019
	88.24	81.97	0.8961
RΔ-MH-LSTM	93.50	89.62	0.9444
SΔ-MH-LSTM (Dice)	97.10	93.88	0.9728
SΔ-MH-LSTM (Cosine)	96.56	93.11	0.9701
SΔ-MH-LSTM (Jacord)	97.08	94.00	0.9727
SΔ-MH-LSTM (Inner)	88.93	86.12	0.8251

In this setup, we use the pre-trained Glove Vector Embedding ($dimension = 100$) instead of random embedding as described in the previous subsection. The original data set is in a raw format. We applied several pre-processing measures like removing numeric and empty texts, converting the rating into binary classes, removing punctuation marks and stop words. Fig. , describes the architecture model we used in this part of the experiments. For the FΔ-MH-LSTM networks, we experimented with various Δ values like $2, 4, 8$ and 10. For the RΔ-MH-LSTM networks, the Δ value range was $[2, 10]$. Likewise, for the SΔ-MH-LSTM networks, we calculated the SM entries using all the similarity coefficients. The number of entries to be calculated for the SM are 49×25.

The accuracies and the AUC-scores for all the algorithms is shown in Table 5.3.

5.5.3 IMDB Movie Review Data Set

In this subsection, we predict the sentiment from the movie reviews from a famous data set IMDB Movie Review data set or also known as Large Movie Review Data Set. The data set consists of highly polar movie reviews. There are at a total of 25000 reviews each for training and testing data set. Overall, there are 88585 unique words but in our experiments we work with a vocabulary size of 500. We again us the Glove Vector Embedding with a dimension of 100 here. The maximum length of the movie review is 500 while the average or mean movie review length is approximately 234 words. The reviews shorter than 500 are padded with zeros in order to make the length of the sequence to be same.

Table 5.4: Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple LSTM networks and MH-LSTM networks. For the FΔ-MH-LSTM networks, we have accuracies and AUC scores for $\Delta = \{5, 20, 40, 60, 80, 100\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN - 2 Layers	84.49	86.99	0.8749
LSTM	91.62	92.16	**0.9487**
FΔ-MH-LSTM ($\Delta = \{5, 20, 40, 60, 80, 100\}$)	86.08	85.54	0.8874
	85.11	84.98	0.8851
	82.07	83.34	0.8319
	80.64	79.99	0.8017
	71.54	74.21	0.7544
	65.12	71.05	0.7341
RΔ-MH-LSTM	86.81	87.77	0.8992
SΔ-MH-LSTM (Dice)	89.12	88.24	0.9110
SΔ-MH-LSTM (Cosine)	90.33	91.76	**0.9356**
SΔ-MH-LSTM (Jacord)	89.15	89.99	0.9247
SΔ-MH-LSTM (Inner)	81.27	76.23	0.7984

FΔ-MH-LSTM networks, we experimented with various Δ values like $5, 20, 40, 60, 80$ and 100. For the RΔ-MH-LSTM networks, the Δ value range

was $[5, 100]$ where the step in Δ we take is 10. Likewise, for the SΔ-MH-LSTM networks, we calculated the SM entries using all the similarity coefficients. The number of entries to be calculated for the SM are 499×250.

The accuracies and the AUC-scores for all the algorithms is shown in Table 5.4.

Remark 5.3: *We now perform the similar set of experiments (as conducted above using the LSTM networks) using the GRU networks and list down the performance. We investigate the steady state error performance, training time profile(in seconds) and the total number of parameters in Appendix D*

Table 5.5: Total number of trainable parameters and training time (in seconds for one epoch) for all the algorithms and data sets

	Twitter US Airline		**Yelp Round-10**		**IMDB Movie Review**	
	Time	**Parameters**	**Time**	**Parmaeters**	**Time**	**Parameters**
FFNN	3914	301247	4128	320061	6452	501245
LSTM	5166	511194	6842	583617	8755	910012
FΔ-MH-LSTM	2691	255597	1461	116724	1823	182002
	1500	127800	397	29780	450	45500
	710	63890	223	14600	230	22750
	-	-	144	9726	145	15167
	-	-	-	-	121	11375
	-	-	-	-	101	9101
RΔ-MH-LSTM	1872	κ	972	κ	1305	κ
SΔ-MH-LSTM (Dice)	1915	$\kappa + 496$	1124	$\kappa + 1225$	2201	$\kappa + 124750$
SΔ-MH-LSTM (Cosine)	2003	$\kappa + 496$	1324	$\kappa + 1225$	2331	$\kappa + 124750$
SΔ-MH-LSTM (Jacord)	2111	$\kappa + 496$	1289	$\kappa + 1225$	1985	$\kappa + 124750$
SΔ-MH-LSTM (Inner)	1754	$\kappa + 496$	1074	$\kappa + 1225$	1804	$\kappa + 124750$

Chapter 6

Conclusions

In the second chapter, we introduce a highly efficient online learning algorithms using the LSTM networks. We add additional yet useful information in the form of covariance into the gating structure of the LSTM networks. We tweak the gating structure of the LSTM networks with both input and output covariance information. We propose three different covariance information based LSTM network models. Furthermore, we then derive the additive gradient based updates for the Co-LSTM networks. Subsequently, in order to alleviate the time and computational complexity as a result of incorporating the covariance information into the LSTM network gating structure, we apply the WMF trick on the CO-LSTM coefficient network weight matrices. We then derive the additive gradient based updates for the WMF based CO-LSTM networks. In the end, we demonstrate the superiority of the Co-LSTM networks through a set of experiments on real-life data sets. In the third chapter, we use network intrusion data set namely ISCX IDS 2012 where we apply the Co-LSTM networks in a both supervised as well as an unsupervised manner to execute network intrusion detection in source and destination payloads.

We then propose a boosted version of the Tree-LSTM networks which we call BBT-LSTM networks in the fourth chapter. We create a balanced tree structure where each LSTM node has binary splittings, i.e., two child LSTM nodes. These

BBT-LSTM networks are a version of the simple N-ary LSTM trees. We added the boosting effect by adding the depth to the tree which significantly enhanced its performance. Since adding depth to the Tree-LSTM networks directly results in a significant increase in time and computational complexity which is not desired especially in an online environment. We reduce the computational complexity of the BBT-LSTM networks by first applying the WMF trick on the BBT-LSTM network weight matrices. We then replace the regular multiplication operator with the energy efficient operator in order to reduce the time complexity. We then combine both energy efficient operator with the WMF in order to reduce both time and computational complexity in one go. We also introduce a slicing operation on the network weight matrices that also significantly reduces the complexity. We then conclude the chapter by illustrating the improved performance of the BBT-LSTM networks as compared with Simple and other Tree-based LSTM networks on real-life data sets.

Finally, in the fifth chapter, we implement another low complexity based version of the LSTM networks based on hopping over the input data sequence. We propose two methods for efficient selection of hopping distance and give its detail which is then followed by computational and time complexity and performance trade-offs. We then illustrate the performance of the MH-LSTM networks by using real-life data sets.

Bibliography

[1] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital signal processing*, vol. 4, no. 4, pp. 207–221, 1994.

[2] I. Ali and Y.-T. Chen, "Design quality and robustness with neural networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1518–1527, 1999.

[3] V. Vovk, "Competitive on-line linear regression," in *Advances in Neural Information Processing Systems*, pp. 364–370, 1998.

[4] A. Krzyzak and T. Linder, "Radial basis function networks and complexity regularization in function learning," in *Advances in neural information processing systems*, pp. 197–203, 1997.

[5] S. Bengio and Y. Bengio, "Taking on the curse of dimensionality in joint distributions using neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 550–557, 2000.

[6] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.

[7] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[8] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[10] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in neural information processing systems*, pp. 190–198, 2013.

[11] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[13] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[14] J. Mazumdar and R. G. Harley, "Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents.," *IEEE Trans. Industrial Electronics*, vol. 55, no. 9, pp. 3484–3491, 2008.

[15] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, vol. 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[16] J. Kivinen and M. K. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *Information and Computation*, vol. 132, no. 1, pp. 1–63, 1997.

[17] O. Kuchaiev and B. Ginsburg, "Factorization tricks for lstm networks," *arXiv preprint arXiv:1703.10722*, 2017.

[18] X. Liu, "Deep recurrent neural network for protein function prediction from sequence," *arXiv preprint arXiv:1701.08318*, 2017.

[19] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[20] Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang, "Enhancing and combining sequential and tree lstm for natural language inference," *arXiv preprint arXiv:1609.06038*, 2016.

[21] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[22] C. E. R. et al., "Delve data sets," *[Online]. Available: http://www.cs.toronto.edu/ delve/data/datasets.html Accessed:*, Oct. 1, 2016.

[23] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework.," *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, 2011.

[24] M. Lichman, "UCI machine learning repository," 2013.

[25] L. Torgo, "Regression data sets."

[26] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[27] D. Hakkani-Tür, G. Tür, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng, and Y.-Y. Wang, "Multi-domain joint semantic frame parsing using bi-directional rnn-lstm.," in *Interspeech*, pp. 715–719, 2016.

[28] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in neural information processing systems*, pp. 3079–3087, 2015.

[29] V. Raghavan, P. Bollmann, and G. S. Jung, "A critical investigation of recall and precision as measures of retrieval system performance," *ACM Transactions on Information Systems (TOIS)*, vol. 7, no. 3, pp. 205–229, 1989.

[30] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240, ACM, 2006.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[33] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.

[34] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[35] H. Tuna, I. Onaran, and A. E. Cetin, "Image description using a multiplier-less operator," *IEEE Signal Processing Letters*, vol. 16, no. 9, pp. 751–753, 2009.

[36] A. Afrasiyabi, B. Nasir, O. Yildiz, F. T. Y. Vural, and A. E. Cetin, "An energy efficient additive neural network," in *Signal Processing and Communications Applications Conference (SIU), 2017 25th*, pp. 1–4, IEEE, 2017.

[37] "Alcoa inc," *[Online], Available,: http:// finance.yahoo.com/quote/AA?ltr=1 Accessed:*, Oct. 1, 2016.

[38] C. Faloutsos and D. W. Oard, "A survey of information retrieval and filtering methods," tech. rep., 1998.

[39] L. Rokach and O. Maimon, "Clustering methods," in *Data mining and knowledge discovery handbook*, pp. 321–352, Springer, 2005.

[40] D. Lin *et al.*, "An information-theoretic definition of similarity.," in *Icml*, vol. 98, pp. 296–304, Citeseer, 1998.

[41] N. Jardine and C. J. van Rijsbergen, "The use of hierarchic clustering in information retrieval," *Information storage and retrieval*, vol. 7, no. 5, pp. 217–240, 1971.

[42] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum, "Information retrieval using a

singular value decomposition model of latent semantic structure," in *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 465–480, ACM, 1988.

[43] "Twitter us airline sentiment data set," *https://www.kaggle.com/yelp-dataset/yelp-dataset/home*.

[44] "Yelp open data set round 10," *https://www.yelp.com/dataset*.

[45] "Imdb movie reviews data set," *https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset*.

Appendix A

Data Sets

We use various real-life data sets in our simulations. The links and references to the data sets are given as follows:

- **Kinematics:** C.E.R. et al., Delve data sets, [Online]. Available: http://www.cs.toronto.edu/delve/data/datasets.html Accessed:, Oct. 1, 2016.
- **Elevators:** J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework.," Journal of Multiple-Valued Logic and Soft Computing, vol. 17, 2011.
- **Protein Tertiary:** M. Lichman, UCI machine learning repository," 2013.
- Puma8NH: L. Torgo, Regression data sets."
- **Alcoa Corporation:** Alcoa inc, [Online], Available,: http:// finance.yahoo.com/quote/AA?ltr=1 Accessed:, Oct. 1, 2016.
- **ISCX IDS 2012:** A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection," computers and security, vol. 31, no. 3, pp. 357-374, 2012.

- **Twitter US Airline Sentiment:** Twitter US Airline sentiment data set," https://www.kaggle.com/yelpdataset/yelp-dataset/home.
- **Yelp Open Round-10:** Yelp open data set round 10," https://www.yelp.com/dataset.
- **IMDB Movie Reviews:** IMDB movie reviews data set," https://www.kaggle.com/iarunava/imdbmovie-reviews-dataset.

Appendix B

Co-GRU Networks

Table B.1: Steady State Error Performance of Simple-GRU and all the models of Co-GRU networks with and without WMF.

Models / Data Sets	**Kinematics**	**Elevators**	**Protein**	**Puma8NH**
Simple GRU	0.4712	0.4899	0.2303	0.1284
Co-GRU (Model 1)	0.4555	4.8978	0.2283	0.1113
Co-GRU (Model 2)	0.4951	4.9280	0.2470	0.1282
Co-GRU (Model 3)	**0.2100**	**4.7321**	**0.1999**	**0.1083**
WMF-Co-GRU (Model 1)	0.4613	4.993	0.2423	0.1201
WMF-Co-GRU (Model 2)	0.4963	4.9420	0.2502	0.1311
WMF-Co-GRU (Model 3)	**0.2166**	**4.7499**	**0.2015**	**0.1095**

Table B.2: Training times (in seconds) for the Simple-GRU and all the models of the Co-GRU networks with and without WMF.

Algorithms	**Time (Kinematics)**	**Time (Elevators)**	**Time (Protein)**	**Time (Puma8NH)**
Simple GRU	51.87	666.61	390.98	410.10
Co-GRU (Model 1)	112.64	154.99	780.43	867.21
Co-GRU (Model 2)	58.01	690.13	423.28	411.11
Co-GRU (Model 3)	62.29	690.27	423.47	400.12
WMF-Co-GRU (Model 1)	105.10	310.12	340.22	337.79
WMF-Co-GRU (Model 2)	52.20	161.34	120.46	157.24
WMF-Co-GRU (Model 3)	55.44	126.49	111.56	140.18

Appendix C

BBT-GRU Networks

Table C.1: Steady State Error performance for all the regression algorithms for Alcoa, Kinematics, Protein Tertiary and Puma8NH data sets

Algorithms / Data Sets	**Alcoa (N=4) rank = 2** $\lceil(\cdot)\rceil = 1$	**Kinematics (N=8) rank = 3** $\lceil(\cdot)\rceil = 2$	**Protein (N=8) rank =3** $\lceil(\cdot)\rceil = 2$	**Puma (N=8) rank = 3** $\lceil(\cdot)\rceil = 1$
GRU	1.6962	0.1317	0.2575	0.1280
N-ary Tree-GRU	1.5195	0.1314	0.2323	0.1058
BBT-GRU	0.9825	0.0931	0.1979	0.0896
ef-BBT-GRU	0.9827	0.0996	0.2010	0.1000
WMF-BBT-GRU	0.9824	0.1014	0.1997	0.1010
ef-WMF-BBT-GRU	0.9825	0.1007	0.2005	0.1010
Sliced-BBT	0.9827	0.1089	0.2001	0.9999
Sliced-BBT-WMF-GRU-1	0.9831	0.1090	0.2011	0.1040
Sliced-BBT-WMF-GRU-2	0.9835	0.1094	0.2106	0.1051

Table C.2: Timing Profile (in seconds) for all the regression algorithms for Alcoa, Kinematics, Protein Tertiary and Puma8NH data sets

Algorithms / Data Sets	**Alcoa (N=4) rank = 2** $\lceil(\cdot)\rceil = 1$	**Kinematics (N=8) rank = 3** $\lceil(\cdot)\rceil = 2$	**Protein (N=8) rank =3** $\lceil(\cdot)\rceil = 2$	**Puma (N=8) rank = 3** $\lceil(\cdot)\rceil = 1$
GRU	21.1230	40.2641	400.0017	437.1111
N-ary Tree-GRU	70.2674	131.7205	2000.3048	2000.5431
BBT-GRU	108.8266	198.1712	2870.1804	3000.1578
ef-BBT-GRU	140.6789	250.7800	3248.0981	3501.1111
WMF-BBT-GRU	88.4564	144.1355	2111.1254	2438.1567
ef-WMF-BBT-GRU	105.1428	170.4854	2477.1878	2669.1518
Sliced-BBT	66.1541	144.2154	2000.0001	2215.4554
Sliced-BBT-WMF-GRU-1	57.2151	131.1112	1999.1256	2254.5454
Sliced-BBT-WMF-GRU-2	30.1697	97.7812	1001.1445	1554.9512

Appendix D

MH-GRU Networks

Table D.1: US Twitter Data Set - Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple GRU networks and MH-GRU networks. For the FΔ-MH-GRU networks, we have accuracies and AUC scores for $\Delta = \{2, 4, 8\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN	78.17	90.57	0.8808
GRU	89.31	96.10	0.9689
FΔ-MH-GRU	83.20	93.12	0.9375
	82.45	92.88	0.9111
	77.58	87.05	0.8634
RΔ-MH-GRU	85.79	93.00	0.9400
SΔ-MH-GRU (Dice)	87.15	94.11	0.9499
SΔ-MH-GRU (Cosine)	89.10	94.47	0.9551
SΔ-MH-GRU (Jacord)	88.88	94.70	0.9567
SΔ-MH-GRU (Inner)	78.18	81.20	0.8650

Table D.2: Yelp Data Set - Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple GRU networks and MH-GRU networks. For the FΔ-MH-GRU networks, we have accuracies and AUC scores for $\Delta = \{2, 4, 8, 10\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN	91.02	85.16	0.8510
GRU	97.91	94.14	0.9789
FΔ-MH-GRU	93.11	90.75	0.9415
	92.06	88.91	0.9401
	90.05	86.25	0.9011
	88.24	81.94	0.8959
RΔ-MH-GRU	93.55	89.64	0.9445
SΔ-MH-GRU (Dice)	96.12	92.88	0.9700
SΔ-MH-GRU (Cosine)	96.56	93.11	0.9701
SΔ-MH-GRU (Jacord)	97.01	94.00	0.9726
SΔ-MH-GRU (Inner)	88.93	86.12	0.8251

Table D.3: IMDB Data Set - Positive and Negative Accuracy and AUC Scores for the feed forward neural network (FFNN), Simple GRU networks and MH-GRU networks. For the FΔ-MH-GRU networks, we have accuracies and AUC scores for $\Delta = \{5, 20, 40, 60, 80, 100\}$

Accuracy / Scores	**Positive Accuracy**	**Negative Acuuracy**	**AUC Score**
FFNN - 2 Layers	84.49	86.99	0.8749
LSTM	91.62	92.16	**0.9487**
FΔ-MH-LSTM ($\Delta = \{5, 20, 40, 60, 80, 100\}$)	86.08	85.54	0.8874
	85.11	84.98	0.8851
	82.07	83.34	0.8319
	80.64	79.99	0.8017
	71.54	74.21	0.7544
	65.12	71.05	0.7341
RΔ-MH-LSTM	86.81	87.77	0.8992
SΔ-MH-LSTM (Dice)	89.12	88.24	0.9110
SΔ-MH-LSTM (Cosine)	90.33	91.76	**0.9356**
SΔ-MH-LSTM (Jacord)	89.15	89.99	0.9247
SΔ-MH-LSTM (Inner)	81.27	76.23	0.7984