Tampere University

Dat Thanh Tran

ATTENTION-AUGMENTED MULTILINEAR NETWORKS FOR TIME-SERIES CLASSIFICATION

Faculty of Information Technology and Communications Sciences
Master of Science Thesis
January 2019

ABSTRACT

Dat Thanh Tran: ATTENTION-AUGMENTED MULTILINEAR NETWORKS FOR TIME-SERIES CLASSIFICATION
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
Major: Machine Learning and Data Engineering
January 2019

Time-series analysis has long been a challenging problem and has been studied extensively over the past decades. In fact, several phenomena possess the dynamic nature of time, with related data collected and expressed in the form of time-series data. While the current development of hardware and software infrastructures provides us a tremendous amount of data to build and validate our models, the noisy and stochastic nature observed in many data modalities still prevent us from having definite solutions. This is especially true for chaotic systems such as the stock market in which the involvement of actors with different goals in the feedback loop leads to complex behaviors.

In this thesis, the author proposes a neural network layer design that incorporates the intuitive idea of bilinear mapping to multivariate time-series, as well as an attention module that enables the layer to automatically calculate and focus on important temporal instances. The contribution of the new design is two-fold. Firstly, the proposed layer is highly interpretable thanks to its ability to quantify the contribution of different instances encoding temporal information. In the post-training and inference phase, the attention quantities can be visualized to highlight the time instances of interest, opening up the opportunity for further analysis. Secondly, the new layer design requires both lower memory and fewer computations compared to the popular attention-based Long-Short-Term-Memory design, which is the state-of-the-art solution.

In order to validate the proposed architecture, the author has conducted experiments on the problem of stock mid-price movement prediction using information available in Limit Order Book. In the algorithmic trading regime, an automated forecasting system is required to be both accurate and efficient since the market operates on nanosecond resolution. Our experimental results demonstrate that the proposed architecture establishes new state-of-the-art forecasting performances in the problem of interest while running much faster than previously proposed solutions.

Keywords:

PREFACE

The work in this thesis was conducted at the Multimedia Research Group, led by Professor Moncef Gabbouj, in the Laboratory of Signal Processing of Tampere University of Technology during 2018. At the time, the author was employed as a research assistant in the same group.

I would like to express my utmost gratitude towards Professor Moncef Gabbouj and Associate Professor Alexandros Iosifidis, who is a Docent of Machine Learning in Tampere University of Technology and an Associate Professor in Aarhus University, Denmark. With tremendous support and mentor from both professors, I have had an excellent opportunity to learn and familiarize myself with scientific work at an early stage. Especially, without the consistent mentoring and encouragement from professor Alexandros Iosifidis since my bachelor degree, I would not have arrived at the outcome of this work as well as my current understanding of many topics in Machine Learning.

Besides, I am grateful for many fruitful discussions on a variety of topics with Dr. Jenni Raitoharju and my colleagues in TC413, namely Anton Muravev, Honglei Zhangh, Dr. Guanqun Cao, Lei Xu and Mohammad Al-sa'd.

Finally, I am forever grateful to my parents and my sister, who have constantly supported me, not only during my academic training but also throughout my whole life.

Tampere, 9th January 2019

Dat Thanh Tran

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ASeq-RNN	Attention Sequence-to-sequence Recurrent Neural Network
BL	Bilinear Layer
BoF	Bag-of-Feature
CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
HFT	High-Frequency Trading
LDA	Linear Discriminant Analysis
LOB	Limit Order Book
LSTM	Long-Short-Term-Memory
MCSDA	Mutlilinear Class-specific Discriminant Analysis
MDA	Multilinear Discriminant Analysis
MLP	Multilayer Perceptron
MTR	Multilinear Time-series Regression
NBoF	Neural Bag-of-Feature
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RR	Ridge Regression
Seq-RNN	Sequence-to-sequence Recurrent Neural Network
SLFN	Single Layer Feedforward Network
SVM	Support Vector Machine
TABL	Temporal Attention augmented Bilinear Layer
TR	Tensor Regression
WMTR	Weighted Multilinear Time-series Regression

1 INTRODUCTION

Time-series classification and prediction have been extensively studied in different application domains. Representative examples include natural language processing [2, 8, 22], medical data analysis [48, 84], human action/behavior analysis [33, 34, 37], meteorology [81], finance and econometrics [1, 5, 38, 41, 42, 43, 44, 45, 46] and generic time-series classification [35, 36, 68]. Due to the complex dynamics of most phenomena, the observed data is highly non-stationary and noisy, representing a limited perspective of the underlying generating process. While significant progress has been made in certain areas such as speech recognition and weather forecasting, in many other areas such as financial market analysis, time-series modeling still faces several obstacles and complexities [80].

The development of both software and hardware infrastructure has enabled the extensive collection of digital footprints, which provides the analysts and practitioners both an opportunity and a challenge. With the emergence of Graphical Progressing Units (GPUs) efficiently designed for matrix operations, we are now able to construct very complex, data-dependent models, which were unimaginable in the past due to infeasible computation and memory requirements. However, the ability to process a massive amount of data and produce a complex model does not guarantee a feasible solution in practice. The reason lies in the fact that most practical applications require real-time or close to real-time processing, which hinders the practical aspect of computationally intensive models. For example, on mobile devices, it is infeasible to run state-of-the-art speech recognition models, which have been produced using a cluster of powerful processing units, not to mention real-time processing.

To reduce the requirement of such complex models during inference, several works have been proposed, under the so-called area network/model compression approach [7]. The general idea of model compression is to reduce the complexity of a pretrained model by pruning model's parameters or computation steps that are thought to marginally affect the performance of the model or lowering the numerical precision (the bit-width) or both. While this approach can provide a solution to meet practical requirements, the performance of compressed models usually drop significantly compared to the original ones. Another approach is to formulate new models that are both computationally efficient and accurate with examples including [31, 71, 72, 74]. While many applications are not intended for usage in low resource environments, improving the computational efficiency is still an important task, especially in critical applications in which the ideal processing time still lies far below what could be achieved with the current infrastructure. For example, modeling for trading purpose, especially in the realm of algorithmic trading, responding time plays a critical role since the market operates on a nanosecond resolution.

During the past decades, several mathematical models have been proposed to extract features from the noisy, nonstationary time-series. For example, stochastic features and market indicators [52, 53] have been widely studied in quantitative analysis in finance. Notable works include autoregressive (AR) [79] and moving average (MA) [65] features, which were later combined as a general framework called autoregressive moving average (ARMA). Its generalization, also known as autoregressive integrated moving average (ARIMA) [4, 69] which incorporates the differencing step to eliminate nonstationarity, is among popular methods for time-series analysis. To ensure tractability, these models are often formulated under many assumptions of the underlying data distribution, leading to poor generalization to future observations [57]. In recent years, the development of machine learning techniques, such as support vector methods [5, 32, 61] and random forest [14], have been applied to time-series forecasting problems to alleviate the dependence on such strong assumptions. As a result, these statistical machines often outperform the traditional ARIMA model in a variety of scenarios [39].

Although the aforementioned machine learning models perform reasonably well, they are not particularly designed to capture the temporal information within time-series data. A class of neural

network architecture called Recurrent Neural Networks (RNN) is specifically designed to extract temporal information from raw sequential data. Although RNNs were developed more than two decades ago [30], they started to become popular in many different application domains [22, 28, 83] only recently thanks to the development in optimization techniques and computation hardware, as well as the availability of large-scale datasets as mentioned previously. Special types of RNNs, such as Long-Short-Term-Memory (LSTM) [29] networks, which were proposed to avoid the gradient vanishing problem in very deep RNN, have become the state-of-the-art in a variety of sequential data prediction problems [22, 23, 28, 83]. Later, Gated Recurrent Unit (GRU) [10] was designed to achieve a similar objective as LSTM with lower computational cost. The beauty of deep neural networks lies in the fact that these architectures allow end-to-end training, which works directly on the raw data representations instead of hand-crafted features. As a result, suitable data-dependent features are automatically extracted, improving the performance and robustness of the whole system.

While deep neural networks in general and recurrent networks, in particular, are biologically inspired and work well in practice, the learned structures are generally difficult to interpret. It has been long known that there exists a visual attention mechanism in human cortex [13, 59, 78] in which visual stimuli from multiple objects compete for neural representation. To further imitate the human learning system, attention mechanisms were developed for several existing neural network architectures to determine the importance of different parts of the input during the learning process [2, 6, 51, 82]. This not only improves the performance of the network being applied to but also contributes to the interpretation of the obtained result by focusing at the specific parts of the input. For example, in image captioning task [82], by incorporating visual attention into a convolutional LSTM architecture, the model explicitly exhibits the correspondence between the generated keywords and the visual subjects. Similar correspondences between the source phrases and the translated phrases are highlighted in attention-based neural machine translation models [2]. While visual and textual data are easy to interpret and intuitive to humans, generic time-series data is more difficult to perceive, making the LSTM models a black box. This hinders the opportunity for post-training analysis. Few attempts have been made to employ attention functionality in LSTM in different time-series forecasting problems such as medical diagnosis [9] and weather forecast [60] or finance [49, 58]. Although adding attention mechanism into recurrent architectures improves the performance and comprehensibility, it incurs a high computational cost for the whole model. As discussed earlier, this impedes the practicality of the model in many cases in which the ability to rapidly train the system and make predictions with continuously large chunks of incoming data plays an important role.

Multivariate time-series data is naturally represented as a second-order tensor. This representation retains the temporal structure encoded in the data, which is essential for a learning model to capture temporal interaction and dependency. By applying a vectorization step, traditional vector-based models fail to capture such temporal cues, leading to inferior performance compared to tensor-based models that work directly on the natural representation of the input. Recent progress in mathematical tools and algorithms pertaining to tensor input have enabled the development of several learning systems based on multilinear projections. For example, popular discriminant and regression criteria were extended for tensor inputs, such as Multilinear Discriminant Analysis (MDA) [47], Multilinear Class-specific Discriminant Analysis (MCSDA) [70] or Tensor Regression (TR) [24]. Regarding neural network formulations, attempts have also been made to learn separate projections for each mode of the input tensors [17, 18]. Motivation to replace linear mapping by the multilinear counterpart stems from the fact that learning separate dependencies between separate modes of the input alleviates the so-called *curse of dimensionality* and greatly reduces the amount of memory and computation required. While a large volume of literature employing multilinear projections was developed for data modalities such as image, video, and text, few works have been dedicated to time-series analysis.

In recent work [75], the author has shown that a linear multivariate regression model could outperform other competing shallow architectures that do not take into account the temporal nature of High-Frequency Trading (HFT) data. While performing reasonably well compared to other shallow architectures, the learning model in [75] has certain short-comings in practice: the analytical solution is computed based on the entire dataset prohibiting its application in an online learning scenario; with a large amount of data, this model clearly underfits the underlying generating process with performance inferior to other models based on deep architectures [76, 77]. In this thesis, the author proposes a neural network layer which incorporates the idea of bilinear pro-

jection in order to learn two separate dependencies for the two modes of multivariate time-series data. Moreover, the author incorporates an attention mechanism that enables the layer to focus on important temporal instances of the input data. By formulation, the layer is differentiable. Thus it can be trained with any mini-batch gradient descend learning algorithm. The contributions of this thesis are as follows:

- A new type of layer architecture is proposed for multivariate time-series data. The proposed layer is designed to leverage the idea of bilinear projection by incorporating an attention mechanism in the temporal mode. The formulation of the attention mechanism directly encourages the competition between neurons representing the same feature at different time instances. The learned model utilizing the proposed layer is highly interpretable by allowing us to look at which specific time instances the learning model attends to.
- The author provides both analytical and experimental evidence that the proposed attention mechanism is highly efficient in terms of computational complexity, allowing development in practical tasks such as financial forecasting.
- Numerical experiments on a large-scale Limit Order Book (LOB) dataset that contains more than 4 million limit orders show that by using a shallow architecture with only two hidden layers, it is possible to outperform by a large margin existing results of deep networks, such as CNN and LSTM, leading to new state-of-the-art performance. Furthermore, the author shows that our proposed attention mechanism can highlight the contribution of different temporal information, opening up opportunities for further analysis of the temporal instances of interest.
- The work described in this thesis has been published in [75] and [73].

The rest of the thesis is organized as follows. In Chapter 2, the theoretical background is presented including a basic notion of supervised learning paradigm and artificial neural network, as well as an overview of the related works focusing on time-series analysis. Chapter 3 presents in detail the main contributions and analysis. Chapter 4 provides empirical validation that was conducted by the author. The chapter starts by describing the task of mid-price movement prediction given LOB data before providing details of experimental procedures, results and quantitative analysis. Chapter 5 concludes the thesis and discusses possible future extensions and analysis.

2 THEORETICAL BACKGROUND

This chapter gives an introduction to the field of machine learning in general and the class of machine learning models called neural networks in particular, as well as a brief description of the related works. The author will first describe the concept of supervised learning and give some practical examples. Then a more technical description is given for the domain of neural networks and related works.

2.1 Machine Learning

According to [50], the concept of machine learning is defined as: "A computer program is said to learn from experience $\mathbf{E}$ with respect to some class of task $\mathbf{T}$ and performance measure $\mathbf{P}$ if its performance at tasks in $\mathbf{T}$, as measured by $\mathbf{P}$, improves with the experience $\mathbf{E}$". In a layman term, the machine learning field concerns with the task of building mathematical models on top of the collected data to make inference later with newly collected data. It is expected that the more data (experience $\mathbf{E}$) used for the modeling process, the better the outcome ($\mathbf{P}$) we would like to achieve for the defined task ($\mathbf{T}$).

While the description might not seem pragmatic, nowadays we see the application of machine learning in several places. In fact, various problems can be posed and solved with a machine learning approach. For example, social networks such as Facebook, have friend/post suggestions or generally content recommendations that suit our interests, or automatic face tagging when someone uploads a photo. Similarly, the search engine can find semantically similar contents to our queries, both in textual and visual formats. In the automobile industry, the development of autonomous vehicles is made possible by automatic recognition and control systems which are built as machine learning solutions.

In order to formulate a problem using machine learning approach, the practitioner must define three main elements: the task $\mathbf{T}$, the performance measure $\mathbf{P}$ and the experience/data $\mathbf{E}$. Let us take as an example the problem of predicting the future closing price of a stock on the following day. The task $\mathbf{T}$ concerns with the question "What is the objective or the target we would like to achieve?". In our example problem, the task $\mathbf{T}$ would be to predict a number (the closing price), given the data in the past and current day. The performance measure $\mathbf{P}$ concerns with the question "Given two solutions $\mathbf{S}_1$ and $\mathbf{S}_2$, how can we quantify which one is better with respect to our task $\mathbf{T}$?". In case of predicting stock prices, we want the predicted prices to be as close as possible to the actual prices. A simple performance measure $\mathbf{P}$ is the absolute error between the predicted price and the actual price. The experience $\mathbf{E}$ concerns with the question "What kind of data we should and could collect in order to solve the task $\mathbf{T}$?". For simple illustration purpose, we assume that the prediction is based on the past and current closing prices and we have access to the closing prices over the period of one thousand days in the past. However, it should be noted that the choice of $\mathbf{E}$ is largely based on the domain knowledge. An economist, for example, would suggest several different market indicators that are known to affect the future stock price.

When approaching a problem using machine learning, we assume that there exists a function that represents the relationship between different entities. For example, in our stock price prediction problem, we can assume that there exists a function $\mathbf{F}$ that can map the five most recent closing prices to the closing prices on the following day. In other words, this function $\mathbf{F}$ takes as input the past and current closing prices $p_{t-4}, \ldots, p_t$, and outputs a single value which is the predicted closing price on the next day $\tilde{p}_{t+1}$. Here the period of five days is chosen arbitrarily in our example, however, in practice, the length of the period is usually defined by a domain expert. While we want $\mathbf{F}$ to generate $\tilde{p}_{t+1}$ as close as possible to the actual value p_{t+1} at any time, whether it is in the past or future, the only experience or the observed data we have is $\mathbf{E}$, the past

prices in 1000 days. Thus, at least we want $\mathbf{F}$ to perform well in $\mathbf{E}$ with respect to $\mathbf{P}$. In other words, we would like $\mathbf{F}$ to minimize the following quantity:

$$\mathcal{L} = \sum_{k=5}^{999} |\tilde{p}_{k+1} - p_{k+1}| = \sum_{k=5}^{999} |\mathbf{F}(\mathbf{x}_k) - p_{k+1}| \tag{2.1}$$

where $\mathbf{x}_k = [p_{k-4}, \ldots, p_k]^T$ and $p_1, \ldots, p_{1000}$ represent the stock closing prices over the period of 1000 days in the experience $\mathbf{E}$. In the machine learning glossary, each pair $(\mathbf{x}_k, p_{k+1})$ is known as a training sample, and the set of stock prices in 1000 days is known as the training set. In addition, the task of finding the function $\mathbf{F}$ that maps the input to a target based on the observed data or the experience $\mathbf{E}$ is known as supervised learning.

The quantity $\mathcal{L}$ in Eq. (2.1) is usually referred to as the empirical loss or the cost function, which measures how far from the ideal solution the function $\mathbf{F}$ is, in terms of $\mathbf{P}$. By formulating the problem in terms of $\mathbf{P}$, $\mathbf{E}$, $\mathcal{L}$, and assuming the existence of $\mathbf{F}$, we have translated the original problem to an optimization problem in which the objective is to minimize $\mathcal{L}$. While the assumption of the existence of $\mathbf{F}$ allows us to define $\mathcal{L}$, without making any further assumption on the form of $\mathbf{F}$, it is impossible to find $\mathbf{F}$ that minimizes $\mathcal{L}$, from the space of all functions. Thus, besides $\mathbf{P}$ and $\mathcal{L}$, specifying the form of $\mathbf{F}$ is an important step in formulating and solving a machine learning problem. As a simple demonstration, we can assume that $\mathbf{F}$ is linear, parameterized by $\mathbf{w}$ and b:

$$\mathbf{F}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{2.2}$$

By limiting $\mathbf{F}$ to have a linear form, we again simplify our problem as finding the coefficients or parameters of the linear function so that $\mathcal{L}$ is minimized. Thus, our optimization problem can be mathematically written as:

$$\underset{\mathbf{w},b}{min} \sum_{k=5}^{999} |\mathbf{w}^T\mathbf{x}_k + b - p_{k+1}| \tag{2.3}$$

At this point, it is not obvious if the global solution of (2.3) exists. However, for most people with elementary algebra knowledge, it is clear that a slight modification to (2.3) will ensure the existence of a global and closed-form solution:

$$\underset{\mathbf{w},b}{min} \sum_{k=5}^{999} \|\mathbf{w}^T\mathbf{x}_k + b - p_{k+1}\|^2 \tag{2.4}$$

Here the absolute error has been replaced by the squared error. It should be noted that since we do not have a general solution to all optimization problems, the choice of the performance measure $\mathbf{P}$ and the form of $\mathbf{F}$ is largely affected by our ability to solve the resulting optimization problem. In the past, $\mathbf{P}$ and $\mathbf{F}$ were often selected so that the resulting optimization problem is convex, thus easily solved by convex optimization techniques. In recent years, with the development in non-convex optimization techniques and powerful computing hardware, more and more complex forms of $\mathbf{F}$ have been used, and $\mathbf{F}$ are often represented under the form of artificial neural networks, or simply neural networks.

2.2 Neural Networks

2.2.1 Definition

As mentioned in the previous section, a neural network represents a function. Originally, the term "neural network" comes from the fact that the design of the elementary computing unit in the function was inspired from a biological neuron. Nowadays, many types or architectures of neural networks are not at all designed to mimic our biological processing units. Thus, in this chapter, the author presents the basics of neural networks in terms of a computation graph.

Using neural networks is an efficient and convenient way to represent a very complex function. The computation is represented in a directed graph with each node representing a computing unit or a sub-function. In a directed edge, the source node indicates the input and the sink node

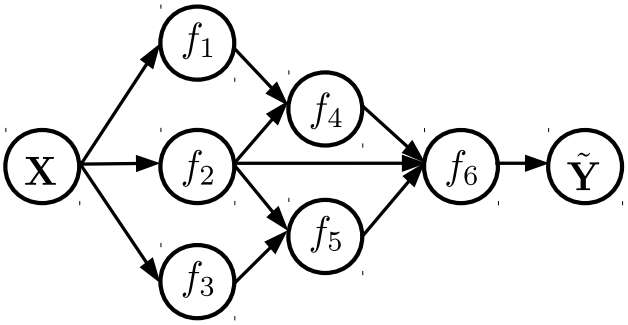

Figure 2.1. *Computation graph of a neural network*

indicates the function that applied to the source node. Let us take as an example the network illustrated in Figure 2.1.

The description of the compound function in Figure 2.1 in written form is the following:

$$\tilde{\mathbf{Y}} = f(\mathbf{X}) \tag{2.5}$$
$$= f_6(f_2; f_4; f_5) \tag{2.6}$$
$$= f_6(f_2(\mathbf{X}); f_4(f_1(\mathbf{X}); f_2(\mathbf{X})); f_5(f_2(\mathbf{X}); f_3(\mathbf{X}))) \tag{2.7}$$

While the neural network in Figure 2.1 is fairly simple, we can see that the description given in Eq. (2.7) is very unintuitive and complicated. Modern neural networks usually consist of hundreds of computing nodes, thus it is almost impossible to be perceived in the written form. In neural network literature, we often encounter the concept of "layer" as a computing unit. A layer usually refers to a set of nodes on the same hierarchy in the graph. For example, in Figure 2.1, f_1, f_2, f_3 can be bagged together and called "first layer"; f_4, f_5 as "second layer" and f_6 as "third layer". However, there is no universal concept of a layer since sometimes f_1, f_2 and f_3 can also be considered as parallel layers which perform different computations. Over the years, certain types of computing units have become prominent due to its efficiencies such as convolution, LSTM or GRU unit, and the community has a more abstract way to describe the structure of the network being used by specifying the number of layers, the types of the layers and so on. Moreover, certain network architectures have become standard designs such as VGG16, VGG19 [63], ResNet18, ResNet101 [26] and so on.

While being omitted in Figure 2.1, each node or computation unit in the graph is parameterized with a set of weights $\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_6}$, and the network as a whole can be seen as having the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_6}\}$. Thus, when the mapping $\mathbf{F}$ is parameterized by a neural network, to minimize the loss function $\mathcal{L}$ is to find the set of parameters $\boldsymbol{\theta}$ that yields the minimum loss value.

2.2.2 Backpropagation Algorithm

Most neural networks represent highly nonlinear functions and we usually have no closed-form solution when optimizing the loss function $\mathcal{L}$. Thus, $\boldsymbol{\theta}$ is usually found through an iterative algorithm in which the parameters are gradually updated using the gradient information. The general idea is that each parameter in the network can be considered as a dimension in the input space of the loss function. The gradient with respect to a parameter reflects the steepest direction in that dimension. Thus, by moving a parameter by a small amount in the opposite direction of

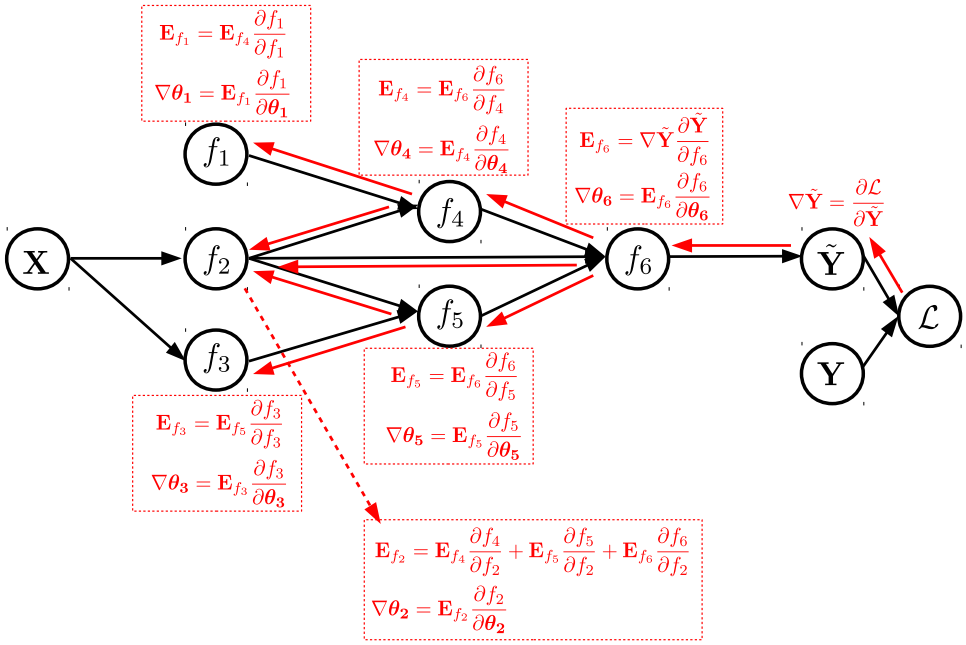

Figure 2.2. *Chain rule in backpropagation. The black arrow indicate the flow of computation of the network and the red arrow indicate the flow of information during backpropagation. Here we denote* $\nabla \boldsymbol{\theta_i} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta_i}}$ *and* $\mathbf{E}_{f_i} = \frac{\partial \mathcal{L}}{\partial f_i}$

the gradient, it is theoretically guaranteed that the move is optimal in order to decrease the loss value. This is known as the delta rule, whose mathematical formula is as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{\theta}_t} \tag{2.8}$$

where α is a hyper-parameter which defines how large is the update step.

These algorithms are generally referred as gradient descent algorithms. In the case when the gradient is estimated only through a subset of the data with no particular order, the algorithm is usually referred as Stochastic Gradient Descent (SGD).

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of weights/parameters update [20]. The method utilizes the chain rule of differentiation to iteratively calculate the gradient for each graph node in the backward order, from the output to the input. Backpropagation is best illustrated through an example. Figure 2.2 illustrate the chain rule applied to the example network in Figure 2.1 with loss function added.

In the example, our objective is to calculate $\partial \mathcal{L}/\partial \boldsymbol{\theta_1}, \ldots, \partial \mathcal{L}/\partial \boldsymbol{\theta_6}$ in order to update the parameters at each iteration using the rule given in Eq. (2.8). Let us denote $\nabla \boldsymbol{\theta_i} = \partial \mathcal{L}/\partial \boldsymbol{\theta_i}$ and $\mathbf{E}_{f_i} = \partial \mathcal{L}/\partial f_i$. Using the chain rule, it is clear that we have the following relation:

$$\nabla \boldsymbol{\theta_i} = \mathbf{E}_{f_i} \frac{\partial f_i}{\partial \boldsymbol{\theta_i}} \tag{2.9}$$

Backpropagation algorithm tells us to go in the backward direction, from the output to the input node. Thus, we start by calculating $\nabla \tilde{\mathbf{Y}}$. Since $\tilde{\mathbf{Y}}$ is a direct input to the loss function $\mathcal{L}$, the calculation of $\nabla \tilde{\mathbf{Y}}$ depends on the form of $\mathcal{L}$. The next step is to calculate $\nabla \boldsymbol{\theta_6} = \partial \mathcal{L}/\partial \boldsymbol{\theta_6}$. According to the chain rule, we have the following relation:

$$\mathbf{E}_{f_6} = \frac{\partial \mathcal{L}}{\partial f_6} \tag{2.10}$$

$$= \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{Y}}} \frac{\partial \tilde{\mathbf{Y}}}{\partial f_6} \tag{2.11}$$

and

$$\nabla \boldsymbol{\theta_6} = \mathbf{E}_{f_6} \frac{\partial f_6}{\partial \boldsymbol{\theta_6}} \tag{2.12}$$

As we can see, the chain rule allows us to express the target gradient as the product of intermediate gradients which are easy to calculate since each element in the denominator is a direct input to the function in the numerator, i.e., f_6 is a direct input of $\tilde{\mathbf{Y}}$ and $\boldsymbol{\theta_6}$ is a direct input of f_6. The quantity $\mathbf{E}_{f_i}$ can be considered as the loss value incurred at f_i, which is calculated by propagating backwards the loss value from $\mathcal{L}$.

Similar calculation steps can be applied to $\nabla \boldsymbol{\theta_5}, \nabla \boldsymbol{\theta_4}, \nabla \boldsymbol{\theta_3}$ and $\nabla \boldsymbol{\theta_1}$. The slightly different case is $\nabla \boldsymbol{\theta_2}$ which receives the gradient information from 3 different nodes (f_4, f_5, f_6) during the backward propagation. The chain rule for $\mathbf{E}_{f_2}$ is the following:

$$\mathbf{E}_{f_2} = \frac{\partial \mathcal{L}}{\partial f_2} \tag{2.13}$$

$$= \frac{\partial \mathcal{L}}{\partial f_4}\frac{\partial f_4}{\partial f_2} + \frac{\partial \mathcal{L}}{\partial f_5}\frac{\partial f_5}{\partial f_2} + \frac{\partial \mathcal{L}}{\partial f_6}\frac{\partial f_6}{\partial f_2} \tag{2.14}$$

$$= \mathbf{E}_{f_4}\frac{\partial f_4}{\partial f_2} + \mathbf{E}_{f_5}\frac{\partial f_5}{\partial f_2} + \mathbf{E}_{f_6}\frac{\partial f_6}{\partial f_2} \tag{2.15}$$

and

$$\nabla \boldsymbol{\theta_2} = \mathbf{E}_{f_2} \frac{\partial f_2}{\partial \boldsymbol{\theta_2}} \tag{2.16}$$

2.2.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of neural network architecture that was proposed specifically for sequential input data. As mentioned in the introduction chapter, many phenomena exhibit a sequential characteristic. For example, in many languages, words in a sentence can only appear in a specific order, so to predict the next possible word, it is essential to look into the sequence of previous words. The term "recurrent" comes from the fact that the same computation step is applied to each element of the sequence.

Figure 2.3 illustrates a simple RNN graph. As we can see from the graph, one special characteristic of RNN is that RNN can operate on an arbitrary length input sequence. The number of computation steps is equal to the length of the input sequence and at each step. Although having been proposed more than two decades ago for sequence-like data, RNNs started to gain interest only recently in time-series analysis community thanks to the advent of GPUs that allows

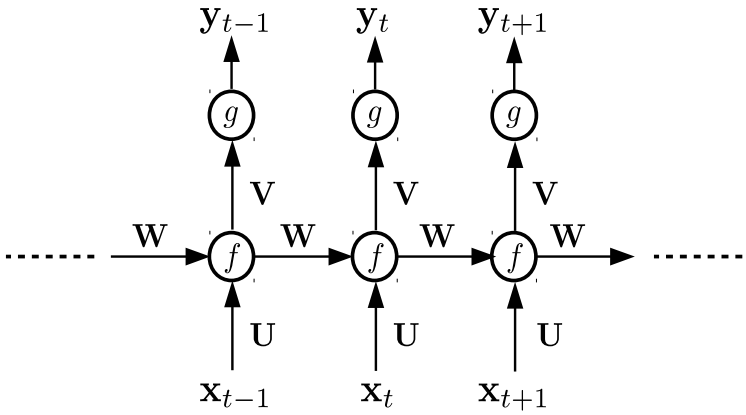

Figure 2.3. *A simple Recurrent Network Graph with output ($\mathbf{y}_t$) at each recurrent step*

the optimization of RNNs in feasible time. The mathematical description of the RNN in Figure 2.3 is the following:

$$\mathbf{s}_t = f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1}) \tag{2.17}$$

$$\mathbf{y}_t = g(\mathbf{V}\mathbf{s}_t) \tag{2.18}$$

where $\mathbf{s}_t$ is called the hidden state at step t, which is calculated from the previous hidden state $\mathbf{s}_{t-1}$ and the input at current step $\mathbf{x}_t$. $\mathbf{s}_0$ is usually set to $\mathbf{0}$. Since a new hidden state is dependent on the previous hidden state, $\mathbf{s}_t$ is usually thought of as the memory units in the network. $\mathbf{y}_t$ is the output at step t, which is calculated from the current hidden state $\mathbf{s}_t$. The nonlinear function f is usually chosen as tanh or Rectified Linear Unit (ReLU). g is usually chosen depending on the output sequence $\mathbf{y}_t$. If $\mathbf{y}_t$ represents the class probability, then g is often chosen as the softmax function.

The example RNN in Figure 2.3 is parameterized by $\mathbf{W}, \mathbf{U}, \mathbf{V}$. When optimizing the RNN using gradient descent approach, we simply follow the backpropagation algorithm presented in the previous section, with the direction going backward in time.

2.3 Related Works

Deep neural networks have been shown to be the state-of-the-art not only in human cognition tasks, such as language and image understanding but also in the prediction of complex time-series data. For example, RNN networks based on LSTM architectures have been used to predict the future rainfall intensity in different geographic areas [81], commodity consumption [60] or to recognize patterns in clinical time-series [48]. In financial data analysis, Deep Belief Networks and Auto-Encoders were used to derive portfolio trading models [27, 62]. In addition, Deep Reinforcement Learning methods are also popular among the class of financial asset trading models [15, 16]. Spatial relations between LOB levels was studied in [64] by a 3-hidden-layer multilayer perceptron (MLP) that models the joint distribution of bid and ask prices. Due to the erratic, noisy nature of stock price movement, many deep neural networks were proposed within a complex forecasting pipeline. For example, in high-frequency LOB data, the authors proposed to normalize the LOB states by the prior date's statistics before feeding them to a CNN [76] or an LSTM network [77]. A more elaborate pipeline consisting of multi-resolution wavelet transform to filter the noisy input series, stacked Auto-Encoder to extract a high-level representation of each stock index and an LSTM network to predict future prices was recently proposed in [3]. Along with popular deep networks, such as CNN, LSTM being applied to time-series forecasting problems, a recently proposed Neural Bag of Feature (NBoF) model was also applied to the problem of stock price movement prediction [55, 56]. The architecture consists of an NBoF layer which compiles histogram representation of the input time-series and a fully-connected layer that classifies the extracted histograms. By learning the parameters of the whole network through Backpropagation algorithm, NBoF was shown to outperform its Bag-of-Feature (BoF) counterpart.

In order to improve both performance and interpretability, attention mechanism was proposed for the recurrent sequence-to-sequence learning problem (ASeq-RNN) [2]. Given a sequence of multivariate inputs $\mathbf{x}_i \in \mathbb{R}^D, 1 \leq i \leq T$ and an associated sequence of outputs $\mathbf{y}_j, 1 \leq j \leq T'$, the Seq-RNN model with attention mechanism learns to generate $\mathbf{y}_j$ from $\mathbf{x}_i$ by using three modules: the encoder, the memory and the decoder. The encoder maps each input $\mathbf{x}_i$ to a hidden state $\mathbf{h}_i^e$ using the nonlinear transformation $\mathbf{h}_i^e = f(\mathbf{x}_i, \mathbf{h}_{i-1}^e)$ coming from a recurrent unit (LSTM or GRU). From the sequence of hidden states generated by the encoder $\mathbf{h}_i^e, i = 1, \ldots, T$, the memory module generates context vectors $\mathbf{c}_j, j = 1, \ldots, T'$ for each output value $\mathbf{y}_j, j = 1, \ldots, T'$. In a normal recurrent model, the context vector is simply selected as the last hidden state $\mathbf{h}_T^e$ while in the attention-based model, the context vectors provide summaries of the input sequence by linearly combining the hidden states $\mathbf{c}_j = \sum_{i=1}^{T} \alpha_{ij} \mathbf{h}_i^e$ through a set of attention weights α_{ij} learned by the following equations:

$$e_{ij} = \mathbf{v}_\alpha^T \tanh(\mathbf{W}_\alpha \mathbf{h}_{j-1}^d + \mathbf{U}_\alpha \mathbf{h}_i^e) \tag{2.19}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{kj})} \tag{2.20}$$

The softmax function in Eq. (2.20) allows the model to produce the context vectors that focus on some time instances of the input sequence and $\mathbf{h}_j^d = g(\mathbf{y}_{j-1}, \mathbf{h}_{j-1}^d, \mathbf{c}_j), j = 1, \ldots, T'$ is the hidden state computed from the recurrent unit in the decoder module. From the hidden state $\mathbf{h}_j^d$ which is based on the previous state $\mathbf{h}_{j-1}^d$, the previous output $\mathbf{y}_{j-1}$ and the current context $\mathbf{c}_j$, the decoder learns to produce the output $\mathbf{y}_j = \mathbf{W}_{out}\mathbf{h}_j^d + \mathbf{b}_{out}$.

Based on the aforementioned attention mechanism, [11] proposed to replace Eq. (2.19) with a modified attention calculation scheme that assumes the existence of pseudo-periods in the given time-series. Their experiments on energy consumption and weather forecast showed that the proposed model learned to attend particular time instances which indicate the pseudo-periods existing in the data. For the future stock price prediction task given current and past prices of multiple stock indices, the authors in [58] developed a recurrent network with two-stage attention mechanism which first focuses on different input series then different time instances. We should note that the above formulations of attention mechanism were proposed for the recurrent structure.

The work described in this thesis can be seen as direct extension of our previous work [75] in which we proposed a regression model based on the bilinear mapping for the mid-price movement classification problem:

$$f(\mathbf{X}) = \mathbf{W}_1 \mathbf{X} \mathbf{w}_2 \tag{2.21}$$

where $\mathbf{X} \in \mathbb{R}^{D \times T}$ is a multivariate time-series containing T temporal steps. $\mathbf{W}_1 \in \mathbb{R}^{3 \times D}$ and $\mathbf{w}_2 \in \mathbb{R}^{T \times 1}$ are the parameters to estimate. To cope with class imbalance problem, [75] proposed a weighted loss function that gives more weights to the minority class:

$$\begin{aligned} J\big(\mathbf{W}_1, \mathbf{w}_2\big) = & \sum_{i=1}^{N} s_i \|\mathbf{W}_1^T \mathbf{X}_i \mathbf{w}_2 - \mathbf{y}_i\|^2 + \\ & \lambda_1 \|\mathbf{W}_1\|^2 + \lambda_2 \|\mathbf{w}_2\|^2 \end{aligned} \tag{2.22}$$

where $\mathbf{y}_i \in \mathbb{R}^C$ is the corresponding target of the ith sample. λ_1 and λ_2 are predefined regularization parameters associated with $\mathbf{W}_1$ and $\mathbf{w}_2$. s_i is the class weight, which is set to be inversely proportional to the number of training samples belonging to the class of sample i, so that errors in smaller classes contribute more to the loss.

In [75], we proposed to solve the optimization in (2.22) through an alternating least-squared approach: at each iterative step, the algorithm alternatively fixes one projection matrix and optimizes the other by calculating the least-squared solution. The procedure is repeated until $\mathbf{W}_1$ and $\mathbf{w}_2$ converge.

By learning two separate mappings that transform the input LOB states to class-membership vector of size 3×1 corresponding to 3 types of movements in mid-price, the regression model in [75] was shown to outperform other shallow classifiers.

Other related works that utilize a bilinear mapping to construct a neural network layer include [17] and [18]. In [17], the authors attempted to incorporate the bilinear mapping into the recurrent structure by processing a block of temporal instances at each recurrent step. Both [17] and [18] focus on medium-scale visual-related tasks such as hand-written digit recognition, image interpolation and reconstruction rather than multivariate time-series data.

3 METHODS

In this chapter, the author introduces the concept of bilinear mapping as a computation unit in the neural network that processes multivariate time-series. Both mathematical formulation and the intuition of bilinear mapping are demonstrated. The chapter then moves on to the detailed description of the temporal attention mechanism. The author concludes the chapter with the complexity analysis of the proposed layer design, in comparison with the existing Attention Sequence-to-Sequence formulation.

3.1 Bilinear Layer

This section starts by providing notations and definitions. Throughout the rest of the thesis, the author denotes scalar values by either lower-case or upper-case character $(a, b, A, B, \dots)$, vectors by lower-case bold-face characters $(\mathbf{x}, \mathbf{y}, \dots)$, matrices by upper-case bold-face characters $(\mathbf{X}, \mathbf{Y}, \dots)$. A matrix $\mathbf{X} \in \mathbb{R}^{D \times T}$ is a second order tensor which has two modes with D and T being the dimension of the first and second mode respectively.

We denote $\mathbf{X}_i \in \mathbb{R}^{D \times T}, i = 1, \dots, N$ the set of N samples, each of which contains a sequence of T observations corresponding to its T columns. So T can be considered as T sampling points in the "time" axis, with the rightmost column representing the most recent information. Here we should note that the term "time" is used in a general sense in which the axis exhibits a sequence-nature.

In time-series forecasting, the time span of the past values (T) is termed as *history* while the time span in the future value (H) that we would like to predict is known as *prediction horizon*. For example, given that the stock prices are sampled every second and the prices from 100 stock in SP100 are collected as the input then the data representation for every minute is $\mathbf{X}_i \in \mathbb{R}^{100 \times 60}$. In that case, the prediction horizon $H = 60$ corresponds to predicting a future value, e.g. a particular stock price, after one minute.

Let us denote by $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_{T_l}] \in \mathbb{R}^{D \times T}$ the input to the Bilinear Layer (BL). The layer transforms an input of size $D \times T$ to a matrix of size $D' \times T'$ by applying the following mapping:

$$\mathbf{Y} = \phi\big(\mathbf{W}_1 \mathbf{X} \mathbf{W}_2 + \mathbf{B}\big) \tag{3.1}$$

where $\mathbf{W}_1 \in \mathbb{R}^{D' \times D}$, $\mathbf{W}_2 \in \mathbb{R}^{T \times T'}$, $\mathbf{B} \in \mathbb{R}^{D' \times T'}$ are the parameters to estimate. $\phi(\cdot)$ is an element-wise nonlinear transformation function, such as ReLU [19] or sigmoid.

As we can see from Eq. (3.1), different than the traditional linear model, BL operates directly on the natural representation of the input time-series, which is a second-order tensor. In a traditional linear model, the input tensor is usually vectorized before applying the mapping. Moreover, compared to linear mapping, one of the obvious advantages of the mapping in Eq. (3.1) is that the number of estimated parameters scales linearly with the dimension of each mode of the input rather than the number of input elements. For a linear layer, transforming a vectorized input of size DT to $D'T'$ requires the estimation of $(DT + 1)D'T'$ parameters (including the bias term), which are much higher than the number of parameters ($DD' + TT' + D'T'$) estimated by the above BL layer.

A more important characteristic of the mapping in Eq. (3.1), when it is applied to time-series data, is that the BL models two dependencies (one for each mode of the input representation), each of which has different semantic meanings. In order to better understand this, denote each column and row of $\mathbf{X}$ as $\mathbf{x}_{c_t} \in \mathbb{R}^D, t = 1, \dots, T$ and $\mathbf{x}_{r_d} \in \mathbb{R}^T, d = 1, \dots, D$, respectively. Given the input time-series $\mathbf{X}$, the t-th column represents D different features or aspects of the underlying process observed at the time instance t, while the d-th row contains the temporal variations of the

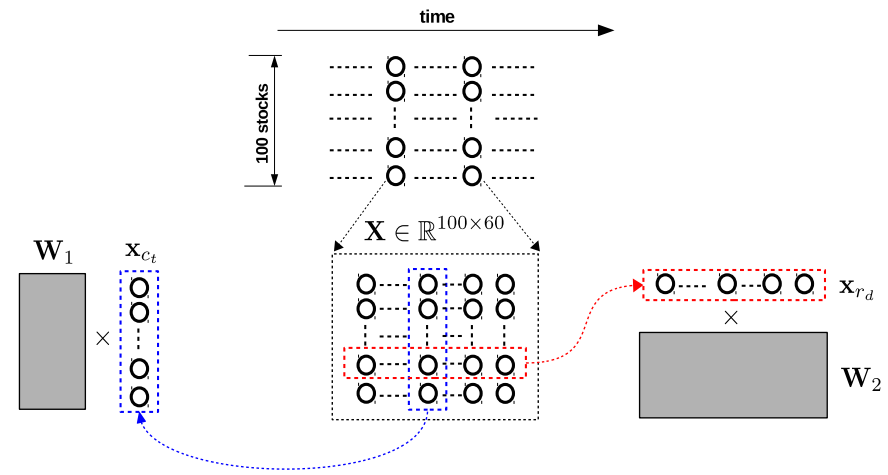

Figure 3.1. *Illustration of Bilinear Layer that operates on input representing* 100 *stock prices in SP100 in an interval of one minute with stock prices sampled at every second.*

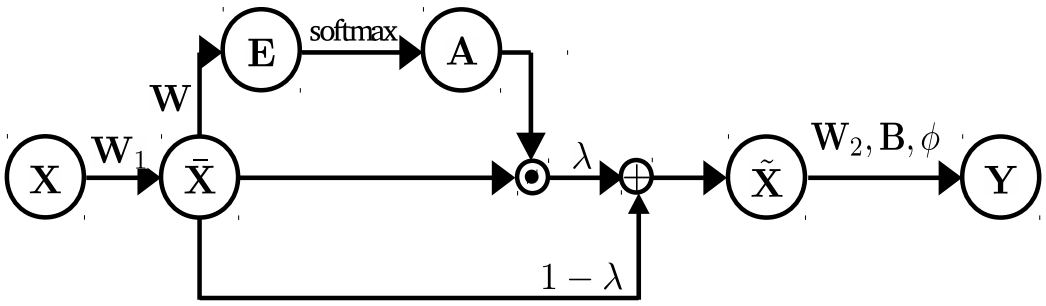

Figure 3.2. *Illustration of the proposed Temporal Attention augmented Bilinear Layer (TABL)*

d-th feature during the past T steps. Since

$$\mathbf{W}_1\mathbf{X} = \left[\mathbf{W}_1\mathbf{x}_{c_1}, \ldots, \mathbf{W}_1\mathbf{x}_{c_T}\right] \tag{3.2}$$

each column $\mathbf{x}_{c_t}$ of $\mathbf{X}$ is linearly transformed by $\mathbf{W}_1$. Thus, the linear relationship between D features is captured by $\mathbf{W}_1$, independent of any time instance. In addition, since

$$\mathbf{X}\mathbf{W}_2 = \begin{bmatrix} (\mathbf{x}_{r_1})^T\mathbf{W}_2 \\ \vdots \\ (\mathbf{x}_{r_D})^T\mathbf{W}_2, \end{bmatrix} \tag{3.3}$$

each row $\mathbf{x}_{r_d}$ of $\mathbf{X}$ is linearly transformed by $\mathbf{W}_2$, which means that the shared temporal progression in the input is modeled by $\mathbf{W}_2$.

Let us take the previous example of $\mathbf{X} \in \mathbb{R}^{100\times 60}$ representing 100 stock prices from SP100 in a minute. The Bilinear Layer captures the general linear relationship between 100 stocks through $\mathbf{W}_1$, and captures the temporal linear fluctuation in SP100 within a minute through $\mathbf{W}_2$. This example is illustrated in Figure 3.1.

3.2 Temporal Attention augmented Bilinear Layer

Although the BL learns separate dependencies along each mode, it is not obvious how a representation at a time instance interacts with other time instances or which time instances are more important in a particular prediction case. For example, by incorporating the position information into the attention calculation scheme, the authors in [11] showed that the learned model only used a particular time instance in the past sequence to predict the future value at a given horizon in the sequence-to-sequence learning task. In order to learn the importance of each time instance in the proposed BL, the author in this thesis proposes the Temporal Attention augmented Bilinear Layer (TABL), a layer design that also maps the input $\mathbf{X} \in \mathbb{R}^{D \times T}$ to the output $\mathbf{Y} \in \mathbb{R}^{D' \times T'}$ with the modified bilinear mapping:

$$\bar{\mathbf{X}} = \mathbf{W}_1 \mathbf{X} \tag{3.4}$$

$$\mathbf{E} = \bar{\mathbf{X}} \mathbf{Q} \tag{3.5}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \tag{3.6}$$

$$\tilde{\mathbf{X}} = \lambda(\bar{\mathbf{X}} \odot \mathbf{A}) + (1 - \lambda)\bar{\mathbf{X}} \tag{3.7}$$

$$\mathbf{Y} = \phi\big(\tilde{\mathbf{X}} \mathbf{W}_2 + \mathbf{B}\big) \tag{3.8}$$

where α_{ij} and e_{ij} denote the element at position (i, j) of $\mathbf{A}$ and $\mathbf{E}$, respectively, $\odot$ denotes the element-wise multiplication operator and $\phi(\cdot)$ is an element-wise nonlinear activation function as in Eq. (3.1).

$\mathbf{W}_1 \in \mathbb{R}^{D' \times D}$, $\mathbf{Q} \in \mathbb{R}^{T \times T}$, $\mathbf{W}_2 \in \mathbb{R}^{T \times T'}$, $\mathbf{B} \in \mathbb{R}^{D' \times T'}$ and λ are the parameters of the proposed TABL. Similar to the aforementioned BL, TABL models two separate dependencies through $\mathbf{W}_1$ and $\mathbf{W}_2$ with the inclusion of the intermediate attention step learned through $\mathbf{Q}$ and λ. The forward pass through TABL consists of 5 steps, which are depicted in Figure 3.2:

- In Eq. (3.4), $\mathbf{W}_1$ is used to transform the representation of each time instance $\mathbf{x}_{c_t}, t = 1, \ldots, T$ of $\mathbf{X}$ (each column) to a new feature space $\mathbb{R}^{D'}$. This models the dependency along the first mode of $\mathbf{X}$ while keeping the temporal order intact.

- The aim of the second step is to learn how important the temporal instances are to each other. This is realized by learning a structured matrix $\mathbf{Q}$ whose diagonal elements are fixed to $1/T$. Let us denote by $\bar{\mathbf{x}}_t \in \mathbb{R}^{D'}$ and $\mathbf{e}_t \in \mathbb{R}^{D'}$ the t-th column of $\bar{\mathbf{X}}$ and $\mathbf{E}$ respectively. From Eq. (3.5), we could see that $\mathbf{e}_t$ is the weighted combination of T temporal instances in the feature space $\mathbb{R}^{D'}$, i.e. T columns of $\bar{\mathbf{X}}$, with the weight of the t-th time instance always equal to $1/T$ since the diagonal elements of $\mathbf{Q}$ are fixed to $1/T$. Thus, element e_{ij} in $\mathbf{E}$ encodes the relative importance of element $\bar{x}_{ij}$ with respect to other $\bar{x}_{ik}, k \neq j$.

- By normalizing the importance scores in $\mathbf{E}$ using the softmax function in Eq. (3.6), the proposed layer pushes many elements to become close to zero, while keeping the values of few of them positive. This process produces the attention mask $\mathbf{A}$.

- The attention mask $\mathbf{A}$ obtained from the third step is used to zero out the effect of unimportant elements in $\mathbb{R}^{D'}$. Instead of applying a *hard* attention mechanism, the learnable scalar λ in Eq. (3.7) allows the model to learn a *soft* attention mechanism. In the early stage of the learning process, the learned features extracted from the previous layer can be noisy and might not be discriminative, thus *hard* attention might mislead the model to unimportant information while *soft* attention could gradually enable the model to learn discriminative features in the early stage, i.e. before selecting the most important ones. Here we should note that λ is constrained to lie in the range $[0, 1]$, i.e. $0 \leq \lambda \leq 1$

- Similar to BL, the final step of the proposed layer performs the temporal mapping through $\mathbf{W}_2$, extracting higher-level representation after the bias shift and nonlinearity transformation.

Generally, the introduction of attention mechanism in the second, third and fourth step of the proposed layer encourages the competition among neurons representing different temporal steps

of the same feature, i.e. competition between elements on the same row of $\bar{\mathbf{X}}$. The competitions are, however, independent for each feature in $\mathbb{R}^{D'}$, i.e. elements of the same column of $\bar{\mathbf{X}}$ do not compete to be represented.

The proposed layer architecture is trained jointly with other layers in the network using the Backpropagation (BP) algorithm. During the backward pass of BP, in order to update the parameters of TABL, the following quantities must be calculated: $\partial\mathcal{L}/\partial\mathbf{W}_1$, $\partial\mathcal{L}/\partial\mathbf{Q}$, $\partial\mathcal{L}/\partial\lambda$, $\partial\mathcal{L}/\partial\mathbf{W}_2$ and $\partial\mathcal{L}/\partial\mathbf{B}$ with $\mathcal{L}$ is the loss function. Derivation of these derivatives is given in the Appendix A.

3.3 Complexity Analysis

Since BL is parameterized by $\mathbf{W}_1 \in \mathbb{R}^{D'\times D}$, $\mathbf{W}_2 \in \mathbb{R}^{T\times T'}$ and $\mathbf{B} \in \mathbb{R}^{D'\times T'}$, the memory complexity of the BL is $O(DD' + TT' + D'T')$. The proposed TABL requires the an additional estimation of $\mathbf{Q} \in \mathbb{R}^{T\times T}$, thus an additional amount of $O(T^2)$ in memory.
Regarding the computational complexity, the BL requires the following computation steps:

- Matrix multiplication $\mathbf{W}_1\mathbf{X}\mathbf{W}_2$ with the cost of $O(D'DT + D'TT')$.
- Bias shift and nonlinear activation with the cost of $O(2D'T')$.

In total, the computational complexity of BL is $O(D'DT + D'TT' + 2D'T')$.

Since TABL possesses the same computation steps as in BL with additional computation for the attention mechanism, the total computational complexity of TABL is $O(D'DT+D'TT'+2D'T'+D'T^2+3D'T)$ with the last two terms contributed from the application the attention mask $\mathbf{A}$.

In order to compare our proposed temporal attention mechanism in bilinear structure with the attention mechanism in a recurrent structure, we estimate the complexity of the attention-based Seq-RNN (ASeq-RNN) proposed in [2] as a reference. Let D' denote the dimension of the hidden units in the encoder, memory and decoder module of ASeq-RNN. In addition, we assume that the input and output sequence have an equal length of T. The total memory and computational complexity of ASeq-RNN are $O(3D'D+11D'^2+11D')$ and $O(11TD'^2+20TD'+4T^2D'+3TD'D+T^2)$ respectively. Details of the estimation are given in the Appendix B.

While configurations of the recurrent and bilinear architecture are not directly comparable, it is still obvious that ASeq-RNN requires far more memory and computation as compared to the proposed TABL. It should be noted that the given complexity of ASeq-RNN is derived based on GRU, which has lower memory and computation complexities compared to LSTM. Variants of ASeq-RNN proposed to time-series data are, however, based on LSTM units [11, 58], making them even more computationally demanding.

4 EVALUATION

In this chapter, the author presents empirical validation of the proposed neural network architecture design on the mid-price movement prediction problem based on a large-scale high-frequency Limit Order Book dataset. Before elaborating on experimental setups and numerical results, the chapter starts by the description of the forecasting task and the dataset used.

4.1 Mid-price Movement Prediction

In stock markets, traders buy and sell stocks through an order-driven system that aggregates all out-standing limit orders in Limit Order Book (LOB). A limit order is a type of order to buy or sell a certain amount of a security at a specified price or better. In a limit order, the trader must specify the type of order (buy/sell), the price and the respective volume (number of stock items he/she wants to trade).

Buy (bid) and sell (ask) limit orders constitute the two sides of the Limit Order Book (LOB), i.e. the bid and ask side. At time t, the best bid price ($p_b^1(t)$) and best ask price ($p_a^1(t)$) are defined as the highest bid and lowest ask prices in the LOB, respectively. When a new limit order arrives, the LOB aggregates and sorts the orders on both sides based on the given prices so that best bid and best ask price are placed at the first level.

If there are limit orders for which the bid price is equal or higher than the lowest ask, i.e. $p_b^1(t) \geq p_a^1(t)$, those orders are immediately executed and removed from the orders book. Different from limit orders, a buy market order is executed immediately with the current best ask price while a sell market order is executed at the current best bid price. An arriving market order is immediately matched with the best available price in the limit order book and a trade occurs, which decreases the depth of the LOB by the number of shares traded. At any time, there are typically multiple levels on both sides of the LOB, and the highest levels are considered to closely reflect current market state. For more information on limit order books, [12, 21] provides a great source of reference information.

The LOB reflects the existing supply and demand for the stock at different price levels. Therefore, based on the availability of LOB data, several analysis and prediction problems can be formulated such as modeling the order flow distribution, the joint distribution of best bid and ask price or casualty analysis of turbulence in the price change. The mid-price at a given time instance is a quantity defined as the mean between the best bid price and best ask price:

$$p_t = \frac{p_a^1(t) + p_b^1(t)}{2} \tag{4.1}$$

This quantity is a virtual price since no trade can take place at this exact price at the same time. Since this quantity lies in the middle of the best bid and best ask price, its movement reflects the dynamics of LOB and the market. Therefore, being able to predict the mid-price movement in the future is of great importance.

4.2 FI-2010 Dataset

To demonstrate the effectiveness of the proposed layer design, the author evaluated the new designs in the task of predicting the future movement of mid-price, given the past bid and ask prices with respective volumes.

The author used the publicly available dataset provided in [54], also known as FI-2010 dataset. The data were collected from 5 different Finnish stocks in NASDAQ Nordic coming from different

industrial sectors. The collection period is from 1st of June to 14th of June 2010, aggregating order data of 10 working days with approximately 4.5 million events. These events can be seen as the sampling points in the time axis. For each event, the prices and volumes of the top 10 orders from each side of the LOB were extracted, producing a 40-dimensional vector representation.

In [54], the authors extract a 144-dimensional feature vector for every non-overlapping block of 10 events, with the first 40 dimensions containing the prices and volumes of the last event in the block, while the rest contain extracted information within the block. This 144-dimensional representation was designed to in the attempt to encapsulate the temporal information contained within the block of 10 events. This feature extraction step is popular among the financial analysts since the community mostly relies upon vector-based model and hand-crafted features.

The feature extraction process results in $453,975$ feature vectors in total. For each feature vector, the dataset includes labels of the mid-price movement (stationary, increase, decrease) in 5 different horizons ($H = 10, 20, 30, 50, 100$) corresponding to the future movements in the next $10, 20, 30, 50, 100$ events.

Since the target of this thesis is to verify the ability of the new layer designs to extract meaningful features and perform the prediction task in an end-to-end fashion, the author only used the raw representation, which is 40-dimensional vector at each time instance representing the prices and volumes of the top 10 levels in the LOB. The database provides different normalized versions of the data such as z-score normalization or min-max normalization. The author conducted all experiments using the provided z-score normalization version.

There exist two experimental setups using FI-2010 dataset. The first setting is the standard anchored forward splits provided by the database which we will refer as Setup1. In Setup1, the dataset is divided into 9 folds based on a daily basis. Specifically, in the k-th fold, data from the first k days is used as the training set while data in the $(k + 1)$-th day is used as the test set with $k = 1, \ldots, 9$. That is, for the 1st fold, data from 1st day is used to train the model while the data collected from the 2nd day is used to test the model; for the 2nd fold, data from the 1st and 2nd day is used to train the model and data from the 3rd day is used to test the model and so on.

The second setting, referred to as Setup2, comes from recent works [76, 77] in which deep network architectures were evaluated. In Setup2, data from the first 7 days were used as the training set while the last 3 days were used as the test set. The empirical results are provided for both settings to allow a wide range of comparisons between existing results in the literature.

4.3 Network Architectures

In order to evaluate the Bilinear Layer (BL) and the Temporal Attention augmented Bilinear Layer (TABL), the author constructed three different baseline network configurations (A,B,C) with 0, 1, and 2 hidden layers that are all Bilinear Layer (BL). Details of the baseline network configurations are shown in Figure 4.1.

The input to all configurations is a matrix of size 40×10 which contains prices and volumes of the top 10 orders from bid and ask side (40 values) spanning over a history of 100 events. Since the feature vector is extracted from a non-overlapping block of 10 events, 100 events correspond to 10 columns in the input matrix. In addition, since the experiments were conducted using only the first 40 dimensions of the given feature vector, the dimension of each column is 40. Here 120×5-BL denotes the Bilinear Layer with output size 120×5. All BLs and TABLs used ReLU as the activation function.

Based on the baseline network configurations, hereby referred to as A(BL), B(BL) and C(BL), the author replaced the last BL classification layer by the proposed attention layer (TABL) to evaluate the effectiveness of attention mechanism. The resulting attention-based configurations are denoted as A(TABL), B(TABL) and C(TABL). Although attention mechanism can be placed in any layer, the author argues that it is more beneficial for the network to attend to high-level representation, which is similar to visual attention mechanism that is applied after applying several convolution layers [82]. In the experiments, no attempt was made to validate all possible positions to apply attention mechanism by simply incorporating it into the last layer.

Table 4.1. *Experimental Results in Setup1*

Models	Accuracy %	Precision %	Recall %	F1 %
Prediction Horizon $H = 10$				
RR[54]	48.00	41.80	43.50	41.00
SLFN[54]	64.30	51.20	36.60	32.70
LDA[75]	63.82	37.93	45.80	36.28
MDA[75]	71.92	44.21	60.07	46.06
MCSDA[70]	83.66	46.11	48.00	46.72
MTR[75]	86.08	51.68	40.81	40.14
WMTR[75]	81.89	46.25	51.29	47.87
BoF[56]	57.59	39.26	51.44	36.28
N-BoF[56]	62.70	42.28	61.41	41.63
A(BL)	44.48	47.56	50.78	43.05
A(TABL)	66.03	56.48	58.09	**56.50**
B(BL)	72.80	65.25	66.92	65.59
B(TABL)	73.62	66.16	68.81	**67.12**
C(BL)	76.82	70.51	72.75	71.33
C(TABL)	78.01	72.03	74.06	**72.84**
Prediction Horizon $H = 50$				
RR[54]	43.90	43.60	43.30	42.70
SLFN[54]	47.30	46.80	46.40	45.90
BoF[56]	50.21	42.56	49.57	39.56
N-BoF[56]	56.52	47.20	58.17	46.15
A(BL)	46.47	54.58	47.83	44.51
A(TABL)	54.61	54.89	53.13	**53.00**
B(BL)	68.09	67.95	67.12	67.16
B(TABL)	69.54	69.12	68.84	**68.84**
C(BL)	74.46	74.20	73.95	73.79
C(TABL)	74.81	74.58	74.27	**74.32**
Prediction Horizon $H = 100$				
RR[54]	42.90	42.90	42.90	41.60
SLFN[54]	47.70	45.30	43.20	41.00
BoF[56]	50.97	42.48	47.84	40.84
N-BoF[56]	56.43	47.27	54.99	46.86
A(BL)	48.90	53.23	45.41	43.40
A(TABL)	51.35	51.37	52.02	**50.66**
B(BL)	66.02	65.78	66.63	65.60
B(TABL)	69.31	68.95	69.41	**68.86**
C(BL)	73.80	73.43	73.40	73.21
C(TABL)	74.07	73.51	73.80	**73.52**

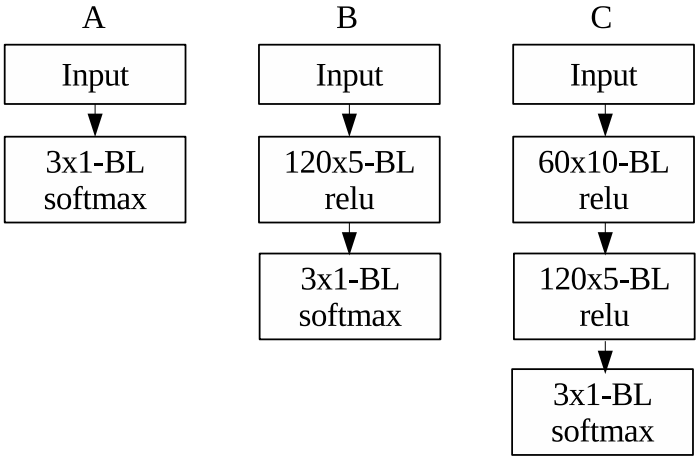

Figure 4.1. *Baseline Network Topologies*

4.4 Experimental Protocols

The following experimental settings were applied to all network configurations mentioned in the previous subsection. Two types of stochastic optimizers were experimented by the author: SGD [67] and Adam [40]. For SGD, the Nesterov momentum was set to 0.9, while for Adam, the exponential decay rates of the first and second moment were fixed to 0.9 and 0.999 respectively. The initial learning rate of both optimizers was set to 0.01 and decreased by the following learning rate schedule $SC = \{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ when the loss in the training set stops decreasing. In total, all configurations were trained for maximum 200 epochs with the mini-batch size of 256 samples.

Regarding regularization techniques, all the networks were regularized with a combination of dropout and max-norm [66], which was shown to improve the generalization capacity of the network. Dropout was applied to the output of all hidden layers with a fixed percentage of 0.1. Max-norm regularizer is a type of weight constraint that enforces an absolute upper bound on the l_2 norm of the incoming weights to a neuron. The maximum norm was selected from the set $\{3.0, 5.0, 7.0\}$. Although weight decay is a popular regularization technique in deep neural network training, the author observes in the initial exploratory experiments that weight decay is not a suitable regularization option when training bilinear structures.

To tackle the problem of class imbalance, the author followed a similar approach proposed in [75] to weight the contribution of each class in the loss function. Since the evaluated network structures output the class-membership probability vector, the following weighted entropy loss function was used:

$$\mathcal{L} = -\sum_{i=1}^{3} \frac{c}{N_i} y_i \log(\tilde{y}_i) \tag{4.2}$$

where N_i, y_i, $\tilde{y}_i$ are the number of samples, true probability and the predicted probability of the i-th class respectively. $c = 10^6$ is a constant used to ensure numerical stability by preventing the loss values being too small when dividing by N_i.

All evaluated networks were initialized with the random initialization scheme proposed in [25] except the attention weight $\mathbf{Q}$ and λ of the TABL layer. Randomly initializing $\mathbf{Q}$ might cause the layer to falsely attend to unimportant input parts at the beginning, leading the network to a bad local minimum. We thus initialized all elements in $\mathbf{Q}$ by a constant equal to $1/T$ with T is the input dimension of the second mode. By initializing $\mathbf{Q}$ with a constant, we ensure that the layer

starts by putting equal focus on all temporal instances. λ was initialized to 0.5 to allow balanced contribution from the original input signals and the masked input signals in the early stage.

4.5 Experimental Results

Following the experimental settings detailed in the previous subsection, the author evaluated the proposed network structures in both Setup1 and Setup2. Besides the performance of the proposed network structures, this section also reports here all available experimental results coming from different models.

For Setup1, this includes Ridge Regression (RR), Single-Layer-Feedforward Network (SLFN), Linear Discriminant Analysis (LDA), Multilinear Discriminant Analysis (MDA), Multilinear Time-series Regression (MTR), Weighted Multilinear Time-series Regression (WMTR) [75], Multilinear Class-specific Discriminant Analysis (MCSDA) [70], Bag-of-Feature (BoF), Neural Bag-of-Feature (N-BoF) [56].

Setup2 includes results from Support Vector Machine (SVM), Multilayer Perceptron (MLP), Convolutional Neural Network (CNN) [76] and LSTM [77].

It should be noted that the results from some models are not entirely comparable since some models (RR, SLFN, LDA, SVM, MLP) can only operate on the vector inputs, thus can only take into account the information contained in the most recent block of 10 events. Other models are specifically designed to operate on the tensor representation of the input data, thus having the privilege of more information in history.

Since the dataset is unbalanced with the majority of samples belonging to the stationary class, the hyper-parameters were tuned based on the average F1 score per class, which is a trade-off between precision and recall, measured on the training set. With the optimal parameter setting, the average performance on the test set over 9 folds is reported in Setup1 while in Setup2, each network configuration is trained 5 times and the average performance on the test set over 5 runs is reported. Besides the main performance metric F1, the author also reports the corresponding accuracy, average precision per class and average recall, also known as sensitivity, per class.

Table 4.1 and 4.2 report the experimental results in Setup1 and Setup2, respectively. As can be seen in Table 4.1, all the competing models in Setup1 belong to the class of shallow architectures with maximum 2 hidden layers (C(BL), C(TABL) and N-BoF). It is clear that all of the bilinear structures outperform other competing models by a large margin on all prediction horizons with the best performances coming from bilinear networks augmented with attention mechanism. Notably, average F1 obtained from the 2 hidden-layer configuration with TABL exceeds the previous best result in Setup1 achieved by WMTR in [75] by nearly 25%. Although NBoF and C(TABL) are both neural network-based architectures with 2 hidden layers, C(TABL) surpasses NBoF by nearly 30% on all horizons. This is not surprising since a regression model based on bilinear projection was shown to even outperform NBoF in [75], indicating that separately learning dependencies in different modes is crucial in time-series LOB data prediction.

While experiments in Setup1 show that the bilinear structure in general and the proposed attention mechanism, in particular, outperform all of the existing models that exploit shallow architectures, experiments in Setup2 establish the comparison between conventional deep neural network architectures and the proposed shallow bilinear architectures. Even with 1 hidden-layer, TABL performs similarly ($H = 20$) or largely better than the previous state-of-the-art results obtained from LSTM network ($H = 10, 50$). Although being deep with 7 hidden layers, the CNN model is greatly inferior to the proposed ones. Here we should note that the CNN proposed in [76] gradually extracts local temporal information by the convolution layers. On the other hand, the evaluated bilinear structures fuse global temporal information from the beginning, i.e. the first layer. The comparison between CNN model and bilinear ones might indicate that the global temporal cues learned in the later stage of CNN (after some convolution layers) lose the discriminative global information existing in the raw data.

Comparing BL and TABL, it is clear that adding the attention mechanism improves the performance of the bilinear networks with an only small increase in the number of parameters. More importantly, the attention mechanism opens up opportunities for further analyzing the contribution of the temporal instances being attended to. This can be seen by looking into the attention mask $\mathbf{A}$. During the training process, each element in $\mathbf{A}$ represents the amount of attention the corresponding element in $\bar{\mathbf{X}}$ receives. In order to observe how each of the 10 events in the input data

Table 4.2. *Experimental Results in Setup2*

Models	Accuracy %	Precision %	Recall %	F1 %
Prediction Horizon $H = 10$				
SVM[77]	-	39.62	44.92	35.88
MLP[77]	-	47.81	60.78	48.27
CNN[76]	-	50.98	65.54	55.21
LSTM[77]	-	60.77	75.92	66.33
A(BL)	29.21	44.08	48.14	29.47
A(TABL)	70.13	56.28	58.26	**56.03**
B(BL)	78.37	67.73	68.89	67.71
B(TABL)	78.91	68.04	71.21	**69.20**
C(BL)	82.52	73.89	76.22	75.01
C(TABL)	84.70	76.95	78.44	**77.63**
Prediction Horizon $H = 20$				
SVM[77]	-	45.08	47.77	43.20
MLP[77]	-	51.33	65.20	51.12
CNN[76]	-	54.79	67.38	59.17
LSTM[77]	-	59.60	70.52	62.37
A(BL)	42.01	47.71	45.38	38.61
A(TABL)	62.54	52.36	50.96	**50.69**
B(BL)	70.33	62.97	60.64	61.02
B(TABL)	70.80	63.14	62.25	**62.22**
C(BL)	72.05	65.04	65.23	64.89
C(TABL)	73.74	67.18	66.94	**66.93**
Prediction Horizon $H = 50$				
SVM[77]	-	46.05	60.30	49.42
MLP[77]	-	55.21	67.14	55.95
CNN[76]	-	55.58	67.12	59.44
LSTM[77]	-	60.03	68.58	61.43
A(BL)	51.92	51.59	50.35	49.58
A(TABL)	60.15	59.05	55.71	**55.87**
B(BL)	72.16	71.28	68.69	69.40
B(TABL)	75.58	74.58	73.09	**73.64**
C(BL)	78.96	77.85	77.04	77.40
C(TABL)	79.87	79.05	77.04	**78.44**

Table 4.3. *Average Computation Time Of State-Of-The-Art Models*

Models	**Forward** (ms)	**Backward** (ms)	**Total** (ms)
A(BL)	0.0038	0.0015	0.0053
A(TABL)	0.0042	0.0042	0.0084
B(BL)	0.0172	0.0174	0.0346
B(TABL)	0.0176	0.0189	0.0365
C(BL)	0.0253	0.0327	0.0580
C(TABL)	0.0254	0.0344	0.0598
CNN	0.0613	0.1100	0.1713
LSTM	0.2291	0.3487	0.5778

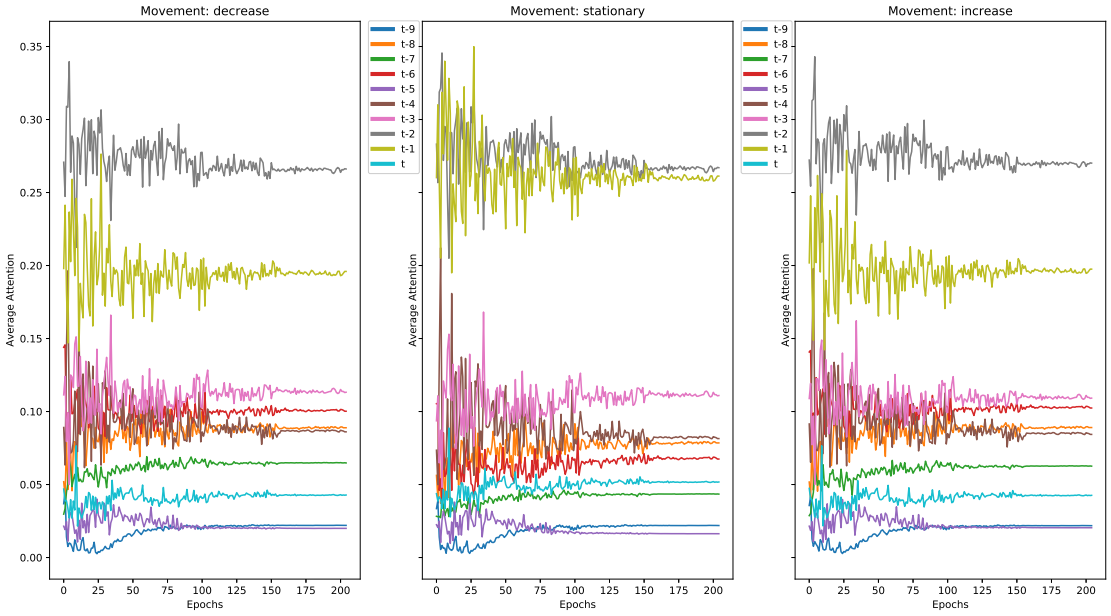

Figure 4.2. *Average attention of 10 temporal instances during training in 3 types of movement: decrease, stationary, increase. Values taken from configuration A(TABL) in Setup2, horizon* $H = 10$

contributes to the decision function, the author analyzed the statistics during the training process of the configuration A(TABL) in Setup2 with horizon $H = 10$.

Figure 4.2 plots the average attention values of each column of $\mathbf{A}$ which correspond to the average attention the model gives to each temporal instance during the training process. The three subplots correspond to attention patterns in three types of mid-price movement. It is clear that the given model focuses more on some events such as the second $(t-1)$, third $(t-2)$ and fourth $(t-3)$ most recent event in all types of movement. While the attention patterns are similar for the decrease and increase class, they are both different when compared to those of the stationary class. This indicates that when the mid-price is going to move from its equilibrium, the model can

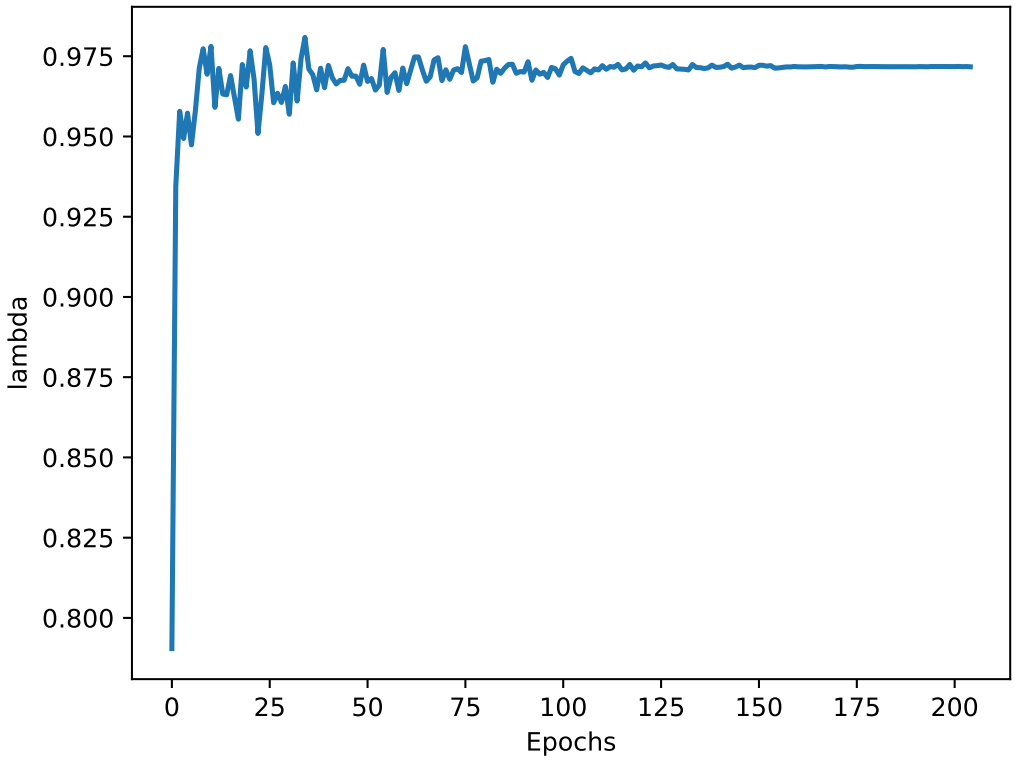

Figure 4.3. *Corresponding λ during training in A(TABL) in Setup2 and horizon $H = 10$*

shift its attention to different events in order to detect the prospective change.

Figure 4.3 shows the corresponding values of λ from the same model after every epoch during the training process. As can be seen from Figure 4.3, λ increases in the first few steps before stabilizing close to 1, which illustrates a *soft* attention behavior achieved by λ described in section 3.2. The insights into the attention patterns and the amount of attention received by each event given by the proposed attention-based layer could facilitate further quantitative analysis such as casualty or pseudo-period analysis.

In order to compare the actual run-time of each model, Table 4.3 reports the average computation time of BL, TABL, CNN [76], LSTM [77] measured on the same machine with CPU core i7-4790 and 32 GB of memory. The second, third and last column shows the average time (in millisecond) taken by the forward pass, backward pass and one training pass of a single sample in the state-of-the-art models. It is obvious that the proposed attention mechanism only increases the computational cost by a relatively small margin. On the contrary, previously proposed deep neural network architectures require around $3\times$ (CNN) and $10\times$ longer to train or generate prediction while having inferior performances compared to the proposed architecture (network configuration C). This points out that the proposed architectures excel previous best models not only in performance but also in efficiency and practicality in applications such as High-Frequency Trading.

5 CONCLUSION

The goal of this thesis work was to develop an efficient neural network architecture design for time-series data, finding the application in financial time-series forecasting. The author has conducted the literature review and studied the advantages and disadvantages of existing works. Bilinear mapping stood out as an intuitive tool to take advantage of the tensor representation of the multivariate time-series. The model constructed by bilinear mapping in our previous work was simple, yet performed relatively well compared to other shallow models. However, it was still inferior compared to deep neural networks.

In this thesis, based on the idea of bilinear mapping, a novel neural network layer design was proposed for time-series data. The proposed layer leverages the idea of bilinear projection and is augmented with a temporal attention mechanism. The author has conducted theoretical analysis on the complexity of the proposed layer in comparison with the existing attention mechanisms in recurrent structures, indicating that the proposed layer possesses much lower memory and computational complexity. In addition, the empirical analysis was conducted in a large-scale Limit Order Book dataset, and showed the effectiveness of the proposed layer design: with only 2 hidden-layers, we can surpass existing state-of-the-art models by a large margin.

The proposed temporal attention mechanism not only improves the performance of the bilinear structure but also enhances its interpretability. The author has illustrated that by looking at the attention values the network gives to different temporal instances during training, we can see distinctive attention patterns in different class labels. Together with proper domain knowledge, the author believes that the quantitative analysis of the attention patterns during the training process could open up opportunities in future research of the patterns of interest.

Since the breadth of the empirical validation provided in this thesis is limited to the problem of predicting mid-price movement using LOB data, extension and validation of the proposed layer design in other application domains involving time-series data will be an interesting future work.

APPENDIX A. TABL DERIVATIVES

In order to calculate the derivatives of TABL, we follow the notation: given $\mathbf{X} \in \mathbb{R}^{I \times J}$ and $\mathbf{X} \in \mathbb{R}^{M \times N}$, $\partial \mathbf{Y} / \partial \mathbf{X}$ is a matrix of size $IJ \times MN$ with element at (ij, mn) equal to $\partial \mathbf{Y}_{ij} / \partial \mathbf{B}_{mn}$. Similarly $\partial \mathcal{L} / \partial \mathbf{X} \in \mathbb{R}^{1 \times MN}$ with $L \in \mathbb{R}, \mathbf{X} \in \mathbb{R}^{M \times N}$. Denote $\mathbf{I}_M \in \mathbb{R}^{M \times M}$ the identity matrix and $\mathbf{1}_{MN} \in \mathbb{R}^{M \times N}$ a matrix with all elements equal to 1. In addition, our derivation heavily uses the following formulas:

$$\frac{\partial(\mathbf{AXB})}{\partial \mathbf{X}} = \mathbf{B}^T \otimes \mathbf{A} \tag{1}$$

$$\begin{aligned} \frac{\partial(\mathbf{A} \odot \mathbf{B})}{\partial \mathbf{C}} &= diag\big(vec(\mathbf{A})\big) \odot \frac{\partial \mathbf{B}}{\partial \mathbf{C}} \\ &+ diag\big(vec(\mathbf{B})\big) \odot \frac{\partial \mathbf{A}}{\partial \mathbf{C}} \end{aligned} \tag{2}$$

where $\otimes$ denotes the Kronecker product, $vec(\mathbf{A})$ denotes the vectorization operator that concatenates columns of $\mathbf{A}$ into one vector and $diag(\mathbf{x})$ denotes the diagonal matrix with the diagonal elements taken from $\mathbf{x}$.

We proceed by calculating the derivate of the left-hand side with respect to every term on the right-hand side from Eq. (3.4) to (3.8):

- From Eq. (3.4)

$$\frac{\partial \bar{\mathbf{X}}}{\partial \mathbf{W}_1} = \frac{\partial(\mathbf{I}_{D'} \mathbf{W}_1 \mathbf{X})}{\partial \mathbf{W}_1} = \mathbf{X}^T \otimes \mathbf{I}_{D'} \tag{3}$$

$$\frac{\partial \bar{\mathbf{X}}}{\partial \mathbf{X}} = \frac{\partial(\mathbf{W}_1 \mathbf{X} \mathbf{I}_T)}{\partial \mathbf{W}_1} = \mathbf{I}_T \otimes \mathbf{W}_1 \tag{4}$$

- From Eq. (3.5)

$$\frac{\partial \mathbf{E}}{\partial \bar{\mathbf{X}}} = \frac{\partial(\mathbf{I}_{D'} \bar{\mathbf{X}} \mathbf{Q})}{\partial \bar{\mathbf{X}}} = \mathbf{Q}^T \otimes \mathbf{I}_{D'} \tag{5}$$

$$\frac{\partial \mathbf{E}}{\partial \mathbf{Q}} = \frac{\partial(\bar{\mathbf{X}} \mathbf{Q} \mathbf{I}_T)}{\partial \bar{\mathbf{X}}} = \mathbf{I}_T \otimes \bar{\mathbf{X}} \tag{6}$$

- From Eq. (3.6) $\partial \mathbf{A} / \partial \mathbf{E}$ is calculated by the following results:

$$\frac{\partial \alpha_{ij}}{\partial e_{ij}} = \alpha_{ij} - \alpha_{ij}^2, \forall i, j \tag{7}$$

$$\frac{\partial \alpha_{ij}}{\partial e_{ip}} = \frac{\alpha_{ij}(1 - \alpha_{ij})}{\sum_{k \neq j, p} \exp(e_{ik})}, \forall p \neq j \tag{8}$$

$$\frac{\partial \alpha_{ij}}{\partial e_{pq}} = 0, \forall p \neq i \tag{9}$$

- From Eq. (3.7)

$$\frac{\partial \tilde{\mathbf{X}}}{\partial \mathbf{A}} = \lambda \frac{\partial (\bar{\mathbf{X}} \odot \mathbf{A})}{\partial \mathbf{A}} = \lambda diag\big(vec(\bar{\mathbf{X}})\big) \tag{10}$$

$$\begin{aligned}\frac{\partial \tilde{\mathbf{X}}}{\partial \bar{\mathbf{X}}} &= \frac{\partial ([\lambda \mathbf{A} + (1-\lambda)\mathbf{1}_{D'T}] \odot \bar{\mathbf{X}})}{\partial \bar{\mathbf{X}}} \\ &= diag\big(vec(\lambda \mathbf{A} + (1-\lambda)\mathbf{1}_{D'T})\big) \\ &\quad + diag\big(vec(\bar{\mathbf{X}})\big) \odot \big(\lambda \frac{\partial \mathbf{A}}{\partial \bar{\mathbf{X}}}\big) \\ &= diag\big(vec(\lambda \mathbf{A} + (1-\lambda)\mathbf{1}_{D'T})\big) \\ &\quad + diag\big(vec(\bar{\mathbf{X}})\big) \odot \big(\frac{\partial \mathbf{A}}{\partial \mathbf{E}} \frac{\partial \mathbf{E}}{\partial \bar{\mathbf{X}}}\big)\end{aligned} \tag{11}$$

$$\frac{\partial \tilde{\mathbf{X}}}{\partial \lambda} = (\bar{\mathbf{X}} \odot \mathbf{A} - \bar{\mathbf{X}}) \tag{12}$$

- In Eq. (3.8), denote $\bar{\mathbf{Y}} = \tilde{\mathbf{X}}\mathbf{W}_2 + \mathbf{B}$, Eq. (3.8) becomes $\mathbf{Y} = \phi(\bar{\mathbf{Y}})$ and we have:

$$\begin{aligned}\frac{\partial \mathbf{Y}}{\partial \tilde{\mathbf{X}}} &= \frac{\partial \mathbf{Y}}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \tilde{\mathbf{X}}} = \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial (\mathbf{I}_{D'} \tilde{\mathbf{X}} \mathbf{W}_2)}{\partial \tilde{\mathbf{X}}} \\ &= \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \mathbf{W}_2^T \otimes \mathbf{I}_{D'}\end{aligned} \tag{13}$$

$$\begin{aligned}\frac{\partial \mathbf{Y}}{\partial \mathbf{W}_2} &= \frac{\partial \mathbf{Y}}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \mathbf{W}_2} = \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial (\tilde{\mathbf{X}} \mathbf{W}_2 \mathbf{I}_T)}{\partial \mathbf{W}_2} \\ &= \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \mathbf{I}_T \otimes \tilde{\mathbf{X}}\end{aligned} \tag{14}$$

$$\begin{aligned}\frac{\partial \mathbf{Y}}{\partial \mathbf{B}} &= \frac{\partial \mathbf{Y}}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \mathbf{B}} = \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial (\mathbf{I}_{D'} \mathbf{B} \mathbf{I}_T)}{\partial \mathbf{B}} \\ &= \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \mathbf{I}_T \otimes \mathbf{I}_{D'}\end{aligned} \tag{15}$$

  where $\partial \phi(\bar{\mathbf{Y}}) / \partial \bar{\mathbf{Y}}$ is the derivative of the element-wise activation function, which depends on the form of the chosen one.

During the backward pass, given $\partial \mathcal{L} / \partial \mathbf{Y}$ and $\partial \phi(\bar{\mathbf{Y}}) / \partial \bar{\mathbf{Y}}$, using chain rules and the above results, the derivatives required in TABL can be calculated as below:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \tilde{\mathbf{X}}} \frac{\partial \tilde{\mathbf{X}}}{\partial \bar{\mathbf{X}}} \frac{\partial \bar{\mathbf{X}}}{\partial \mathbf{W}_1} \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Q}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \tilde{\mathbf{X}}} \frac{\partial \tilde{\mathbf{X}}}{\partial \mathbf{A}} \frac{\partial \mathbf{A}}{\partial \mathbf{E}} \frac{\partial \mathbf{E}}{\partial \mathbf{Q}} \tag{17}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \frac{\partial \phi(\bar{\mathbf{Y}})}{\partial \bar{\mathbf{Y}}} \frac{\partial \bar{\mathbf{Y}}}{\partial \tilde{\mathbf{X}}} \frac{\partial \tilde{\mathbf{X}}}{\partial \lambda} \tag{18}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}_2} \tag{19}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{B}} \tag{20}$$

APPENDIX B. COMPLEXITY OF ATTENTION-BASED RNN

The attention-based sequence to sequence learning proposed in [2] comprises of the following modules:

- Encoder

$$\mathbf{z}_i^e = \sigma\big(\mathbf{W}_z^e \mathbf{x}_i + \mathbf{U}_z^e \mathbf{h}_{i-1}^e + \mathbf{b}_z^e\big) \tag{21}$$
$$\mathbf{r}_i^e = \sigma\big(\mathbf{W}_r^e \mathbf{x}_i + \mathbf{U}_r^e \mathbf{h}_{i-1}^e + \mathbf{b}_r^e\big) \tag{22}$$
$$\tilde{\mathbf{h}}_i^e = \tanh\big(\mathbf{W}^e \mathbf{x}_i + \mathbf{U}^e(\mathbf{r}_i^e \odot \mathbf{h}_{i-1}^e) + \mathbf{b}^e\big) \tag{23}$$
$$\mathbf{h}_i^e = (1 - \mathbf{z}_i^e) \odot \mathbf{h}_{i-1}^e + \mathbf{z}_i^e \odot \tilde{\mathbf{h}}_i^e \tag{24}$$
$$\tag{25}$$

- Memory

$$e_{ij} = \mathbf{v}_\alpha^T \tanh(\mathbf{W}_\alpha \mathbf{h}_{j-1}^d + \mathbf{U}_\alpha \mathbf{h}_i^e) \tag{26}$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{kj})} \tag{27}$$
$$\mathbf{c}_j = \sum_{i=1}^{T} \alpha_{ij} \mathbf{h}_i^e \tag{28}$$

- Decoder

$$\mathbf{z}_j^d = \sigma\big(\mathbf{w}_z^d y_{j-1} + \mathbf{U}_z^d \mathbf{h}_{j-1}^d + \mathbf{C}_z \mathbf{c}_j + \mathbf{b}_z^d\big) \tag{29}$$
$$\mathbf{r}_j^d = \sigma\big(\mathbf{w}_r^d y_{j-1} + \mathbf{U}_r^d \mathbf{h}_{j-1}^d + \mathbf{C}_r \mathbf{c}_j + \mathbf{b}_r^d\big) \tag{30}$$
$$\tilde{\mathbf{h}}_j^d = \tanh\big(\mathbf{w}^d y_{j-1} + \mathbf{U}^d(\mathbf{r}_j^d \odot \mathbf{h}_{j-1}^d) + \mathbf{C}\mathbf{c}_j + \mathbf{b}^d\big) \tag{31}$$
$$\mathbf{h}_j^d = (1 - \mathbf{z}_j^d) \odot \mathbf{h}_{j-1}^d + \mathbf{z}_j^d \odot \tilde{\mathbf{h}}_j^d \tag{32}$$
$$y_j = \mathbf{w}_{out}^T \mathbf{h}_j^d + b_{out} \tag{33}$$

where $i = 1, \ldots, T$ and $j = 1, \ldots, T$ denote the index in input and output sequence respectively, which we assume having equal length. In order to generate sequence of prediction rather than probability of a word in a dictionary, we use Eq. (33) similar to [11]. To simplify the estimation, let the number of hidden units in the encoder, memory and decoder module equal to D', i.e. $\mathbf{h}_i^e, \mathbf{h}_j^d, \mathbf{v}_\alpha \in \mathbb{R}^{D'}$, and the output y_j is a scalar.

The encoder module estimates the following parameters: $\mathbf{W}^e, \mathbf{W}_r^e, \mathbf{W}_z^e \in \mathbb{R}^{D' \times D}$, $\mathbf{U}^e, \mathbf{U}_r^e, \mathbf{U}_z^e \in \mathbb{R}^{D' \times D'}$, $\mathbf{b}^e, \mathbf{b}_r^e, \mathbf{b}_z^e \in \mathbb{R}^{D'}$, which result in $O(3D'D + 3D'^2 + 3D')$ memory and $O(T(3D'D + 3D'^2 + 8D'))$ computation.

The memory module estimates the following parameters: $\mathbf{v}_\alpha \in \mathbb{R}^{D'}$, $\mathbf{W}_\alpha, \mathbf{U}_\alpha \in \mathbb{R}^{D' \times D'}$, which cost $O(2D'^2 + D')$ memory and $O(2D'^2T + 4T^2D' + T^2)$ computation.

The decoder module estimates the following parameters: $\mathbf{U}_r^d, \mathbf{U}_z^d, \mathbf{U}^d, \mathbf{C}_z, \mathbf{C}_r, \mathbf{C} \in \mathbb{R}^{D' \times D'}$, $\mathbf{w}_r^d, \mathbf{w}_z^d, \mathbf{w}^d, \mathbf{b}_z^d, \mathbf{b}_r^d, \mathbf{b}^d, \mathbf{w}_{out} \in \mathbb{R}^{D'}$, which result in $O(6D'^2 + 7D')$ memory and $O(T(12D' + 6D'^2))$.

In total, the attention model requires $O(3D'D + 11D'^2 + 11D')$ memory and $O(11TD'^2 + 20TD' + 4T^2D' + 3TD'D + T^2)$ computation.

REFERENCES

[1] E. M. Azoff. *Neural network time series forecasting of financial markets*. John Wiley & Sons, Inc., 1994.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In: *arXiv preprint arXiv:1409.0473* (2014).

[3] W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. In: *PloS one* 12.7 (2017), e0180944.

[4] G. E. Box, G. M. Jenkins, and J. F. MacGregor. Some recent advances in forecasting and control. In: *Applied Statistics* (1974), 158–179.

[5] L.-J. Cao and F. E. H. Tay. Support vector machine with adaptive parameters in financial time series forecasting. In: *IEEE Transactions on neural networks* 14.6 (2003), 1506–1518.

[6] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia. ABC-CNN: An attention based convolutional neural network for visual question answering. In: *arXiv preprint arXiv:1511.05960* (2015).

[7] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. In: *arXiv preprint arXiv:1710.09282* (2017).

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *arXiv preprint arXiv:1406.1078* (2014).

[9] E. Choi, M. T. Bahadori, J. Sun, J. Kulas, A. Schuetz, and W. Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In: *Advances in Neural Information Processing Systems*. 2016, 3504–3512.

[10] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In: *arXiv preprint arXiv:1412.3555* (2014).

[11] Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Aıt-Bachir, and V. Strijov. Position-Based Content Attention for Time Series Forecasting with Sequence-to-Sequence RNNs. In: *International Conference on Neural Information Processing*. Springer. 2017, 533–544.

[12] R. Cont. Statistical modeling of high-frequency financial data. In: *IEEE Signal Processing Magazine* 28.5 (2011), 16–25.

[13] M. Corbetta and G. L. Shulman. Control of goal-directed and stimulus-driven attention in the brain. In: *Nature reviews neuroscience* 3.3 (2002), 201–215.

[14] G. G. Creamer and Y. Freund. Predicting performance and quantifying corporate governance risk for latin american adrs and banks. In: (2004).

[15] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. In: *IEEE transactions on neural networks and learning systems* 28.3 (2017), 653–664.

[16] Y. Deng, Y. Kong, F. Bao, and Q. Dai. Sparse coding-inspired optimal trading system for HFT industry. In: *IEEE Transactions on Industrial Informatics* 11.2 (2015), 467–475.

[17] K. Do, T. Tran, and S. Venkatesh. Matrix-centric Neural Networks. In: *arXiv preprint arXiv:1703.01454* (2017).

[18] J. Gao, Y. Guo, and Z. Wang. Matrix neural networks. In: *International Symposium on Neural Networks*. Springer. 2017, 313–320.

[19] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, 315–323.

[20] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[21] M. D. Gould, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison. Limit order books. In: *Quantitative Finance* 13.11 (2013), 1709–1742.

[22] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE. 2013, 6645–6649.

[23] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. In: *IEEE transactions on neural networks and learning systems* (2017).

[24] W. Guo, I. Kotsia, and I. Patras. Tensor learning for regression. In: *IEEE Transactions on Image Processing* 21.2 (2012), 816–827.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. 2015, 1026–1034.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778.

[27] J. Heaton, N. Polson, and J. Witte. Deep Portfolio Theory. In: *arXiv preprint arXiv:1605.07230* (2016).

[28] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. In: *IEEE Signal Processing Magazine* 29.6 (2012), 82–97.

[29] S. Hochreiter and J. Schmidhuber. Long short-term memory. In: *Neural computation* 9.8 (1997), 1735–1780.

[30] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In: *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*. World Scientific, 1987, 411–415.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In: *arXiv preprint arXiv:1704.04861* (2017).

[32] A. Iosifidis and M. Gabbouj. Multi-class support vector machine classifiers using intrinsic and penalty graphs. In: *Pattern Recognition* 55 (2016), 231–246.

[33] A. Iosifidis, A. Tefas, and I. Pitas. Activity based Person Identification using Fuzzy Representation and Discriminant Learning. In: *IEEE Transactions on Information Forensics and Security* 7.2 (2012), 530–2012.

[34] A. Iosifidis, A. Tefas, and I. Pitas. View-invariant action recognition based on Artificial Neural Networks. In: *IEEE Transactions on Neural Networks and Learning Systems* 23.3 (2012), 412–424.

[35] A. Iosifidis, A. Tefas, and I. Pitas. Multidimensional Sequence Classification based on Fuzzy Distances and Discriminant Analysis. In: *IEEE Transactions on Knowledge and Data Engineering* 93.6 (2013), 1445–1457.

[36] A. Iosifidis, A. Tefas, and I. Pitas. Discriminant Bag of Words based Representation for Human Action Recognition. In: *Pattern Recognition Letters* 49 (2014), 185–192.

[37] A. Iosifidis, A. Tefas, and I. Pitas. Class-specific Reference Discriminant Analysis with application in Human Behavior Analysis. In: *IEEE Transactions on Human-Machine Systems* 45.3 (2015), 315–326.

[38] I. Kaastra and M. Boyd. Designing a neural network for forecasting financial and economic time series. In: *Neurocomputing* 10.3 (1996), 215–236.

[39] M. J. Kane, N. Price, M. Scotch, and P. Rabinowitz. Comparison of ARIMA and Random Forest time series models for prediction of avian influenza H5N1 outbreaks. In: *BMC bioinformatics* 15.1 (2014), 276.

[40] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014).

[41] S. Lahmiri. A variational mode decompoisition approach for analysis and forecasting of economic and financial time series. In: *Expert Systems with Applications* 55 (2016), 268–273.

[42] S. Lahmiri. Interest rate next-day variation prediction based on hybrid feedforward neural network, particle swarm optimization, and multiresolution techniques. In: *Physica A: Statistical Mechanics and its Applications* 444 (2016), 388–396.

[43] S. Lahmiri. Intraday stock price forecasting based on variational mode decomposition. In: *Journal of Computational Science* 12 (2016), 23–27.

[44] S. Lahmiri. Comparing variational and empirical mode decomposition in forecasting day-ahead energy prices. In: *IEEE Systems Journal* 11.3 (2017), 1907–1910.

[45] S. Lahmiri. Modeling and predicting historical volatility in exchange rate markets. In: *Physica A: Statistical Mechanics and its Applications* 471 (2017), 387–395.

[46] S. Lahmiri. Minute-ahead stock price forecasting based on singular spectrum analysis and support vector regression. In: *Applied Mathematics and Computation* 320 (2018), 444–451.

[47] Q. Li and D. Schonfeld. Multilinear discriminant analysis for higher-order tensor data classification. In: *IEEE transactions on pattern analysis and machine intelligence* 36.12 (2014), 2524–2537.

[48] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell. Learning to diagnose with LSTM recurrent neural networks. In: *arXiv preprint arXiv:1511.03677* (2015).

[49] M. Mäkinen, A. Iosifidis, M. Gabbouj, and J. Kanniainen. Predicting Jump Arrivals in Stock Prices Using Neural Networks with Limit Order Book Data. In: (2018).

[50] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[51] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In: *Advances in neural information processing systems*. 2014, 2204–2212.

[52] J. J. Murphy. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[53] C. J. Neely, D. E. Rapach, J. Tu, and G. Zhou. Forecasting the equity risk premium: the role of technical indicators. In: *Management Science* 60.7 (2014), 1772–1791.

[54] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Benchmark Dataset for Mid-Price Prediction of Limit Order Book data. In: *arXiv preprint arXiv:1705.03233* (2017).

[55] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Temporal Bag-of-Features Learning for Predicting Mid Price Movements Using High Frequency Limit Order Book Data. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2018).

[56] N. Passalis, A. Tsantekidis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Time-series classification using neural bag-of-features. In: *Signal Processing Conference (EUSIPCO), 2017 25th European*. IEEE. 2017, 301–305.

[57] J. M. Poterba and L. H. Summers. Mean reversion in stock prices: Evidence and implications. In: *Journal of financial economics* 22.1 (1988), 27–59.

[58] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. Cottrell. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. In: *arXiv preprint arXiv:1704.02971* (2017).

[59] R. A. Rensink. The dynamic representation of scenes. In: *Visual cognition* 7.1-3 (2000), 17–42.

[60] M. Riemer, A. Vempaty, F. Calmon, F. Heath, R. Hull, and E. Khabiri. Correcting forecasts with multifactor neural attention. In: *International Conference on Machine Learning*. 2016, 3010–3019.

[61] N. I. Sapankevych and R. Sankar. Time series prediction using support vector machines: a survey. In: *IEEE Computational Intelligence Magazine* 4.2 (2009).

[62] A. Sharang and C. Rao. Using machine learning for medium frequency derivative portfolio trading. In: *arXiv preprint arXiv:1512.06228* (2015).

[63] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In: *arXiv preprint arXiv:1409.1556* (2014).

[64] J. Sirignano. Deep Learning for Limit Order Books. In: *ArXiv e-prints* (Jan. 2016). arXiv: 1601.01987 [q-fin.TR].

[65] E. Slutzky. The summation of random causes as the source of cyclic processes. In: *Econometrica: Journal of the Econometric Society* (1937), 105–146.

[66] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. In: *Journal of machine learning research* 15.1 (2014), 1929–1958.

[67] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In: *International conference on machine learning*. 2013, 1139–1147.

[68] S. B. Taieb and A. F. Atiya. A bias and variance analysis for multistep-ahead time series forecasting. In: *IEEE transactions on neural networks and learning systems* 27.1 (2016), 62–76.

[69] G. C. Tiao and G. E. Box. Modeling multiple time series with applications. In: *journal of the American Statistical Association* 76.376 (1981), 802–816.

[70] D. T. Tran, M. Gabbouj, and A. Iosifidis. Multilinear Class-Specific Discriminant Analysis. In: *Pattern Recognition Letters* (2017).

[71] D. T. Tran and A. Iosifidis. Heterogeneous Multilayer Generalized Operational Perceptron. In: *arXiv preprint arXiv:1804.05093* (2018).

[72] D. T. Tran, A. Iosifidis, and M. Gabbouj. Improving efficiency in convolutional neural networks with multilinear filters. In: *Neural Networks* (2018).

[73] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj. Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis. In: *IEEE transactions on neural networks and learning systems* (2018).

[74] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis. Progressive Operational Perceptron with Memory. In: *arXiv preprint arXiv:1808.06377* (2018).

[75] D. T. Tran, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Tensor representation in high-frequency financial data for price change prediction. In: *IEEE Symposium Series on Computational Intelligence (SSCI)* (2017).

[76] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In: *Business Informatics (CBI), 2017 IEEE 19th Conference on*. Vol. 1. IEEE. 2017, 7–12.

[77] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Using deep learning to detect price change indications in financial markets. In: *Signal Processing Conference (EUSIPCO), 2017 25th European*. IEEE. 2017, 2511–2515.

[78] S. K. Ungerleider and G. Leslie. Mechanisms of visual attention in the human cortex. In: *Annual review of neuroscience* 23.1 (2000), 315–341.

[79] G. Walker. On periodicity in series of related terms. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 131.818 (1931), 518–532.

[80] B. Wang, H. Huang, and X. Wang. A novel text mining approach to financial time series forecasting. In: *Neurocomputing* 83 (2012), 136–145.

[81] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In: *Advances in neural information processing systems*. 2015, 802–810.

[82] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In: *International Conference on Machine Learning*. 2015, 2048–2057.

[83] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 4694–4702.

[84] M. Zabihi, A. B. Rad, S. Kiranyaz, M. Gabbouj, and A. K. Katsaggelos. Heart sound anomaly and quality detection using ensemble of neural networks without segmentation. In: *Computing in Cardiology Conference (CinC), 2016*. IEEE. 2016, 613–616.