FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

DEEP LEARNING FOR LIMIT ORDER BOOK TRADING AND MID-PRICE MOVEMENT PREDICTION

By

HETING YAN

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2020

Heting Yan defended this dissertation on February 11, 2020.
The members of the supervisory committee were:

Alec N. Kercheval
Professor Directing Dissertation

Jinfeng Zhang
University Representative

Giray Ökten
Committee Member

Linjiong Zhu
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Deep learning has been widely used to predict price movements from the limit order book. In this paper, we design a consistently profitable trading system for predicting the bid-ask spread crossing. Our trading experiment is done on a limit order book sample dataset from Lobsterdata.com. We improve the daily return rate by 1000% comparing to a paper [1] working on deep learning trading with limit order book dataset from London stock exchange. We introduce a regular mid-price movement model and a benchmark mid-price movement model on the limit order book data from Lobsterdata.com and a benchmark dataset [15]. Our benchmark mid-price movement model improves the f1 score on the benchmark dataset and fixes the bias comparing to the model in the paper [16].

CHAPTER 1

INTRODUCTION

1.1 Motivation

Nowadays, financial trading occurs at places like the New York Stock Exchange (NYSE) and various NASDAQ exchanges. Whenever a trader makes an action, there is a record in the Limit Order Book (LOB) [24] [25] [26] [27] [28] [29] [6]. The Limit Order Book is a collection of data that can represent the state of the market and how it changes. The trading actions are driven by limit orders submitted by an experienced investor or a computer algorithm. The advantage of a computer algorithm is that it is fast, and it can work continuously. Meanwhile, the financial market is changing all the time and is affected by many factors, such as politics, weather, scandals, and advertising. In order to take advantage of such circumstances, the algorithm needs to observe the market and understand how to conduct right trades to maximize the profit with low risk. Therefore, the algorithm not only needs to be fast but also smart. This is why we bring in artificial intelligence.

As the software industry is booming, artificial intelligence has been widely used in many areas [18]. Yahoo! uses artificial intelligence on their digital advertising platform to make more revenue [19]. Netflix uses artificial intelligence to give users personalized recommendations for movies [20]. Amazon Alexa is a virtual shopping assistant who uses artificial intelligence to chat with users [21]. Google uses artificial intelligence to beat the world's best Go player [22]. Uber uses artificial intelligence to build self-driving car [23]. Artificial intelligence can formulate a complex trading algorithm to capture important information which cannot be discovered by humans [33] [32] [31]. Even though the training may take a long time, artificial intelligence can run fast when put it into use for predicting. It is very likely that artificial intelligence can outperform human traders [38] [37] [36] [35] [34].

Convolutional neural network (CNN) has been widely used in all kinds of time series forecasting [46] [47] [48] [49] [50]. Convolutional neural networks have an outstanding ability to exploit the spatial pattern in image data. By treating multidimensional time series as an image, convolutional neural networks can effectively extract the underlying information from it [7] [45]. The limit order book data contains ask price, ask volume, bid price and bid volume at different depth. The changes between prices and volumes at different depth over time can represent the traders' minds. This is ideal for convolutional neural networks. Another advantage of convolutional neural networks is that it is fast at training. The convolution operation in the convolution neural network dramatically reduces the number of parameters compared to a simple fully connected neural network [44] [43] [42]. It also makes the convolution neural network fast to train. The training speed can be dramatically boosted by parallel computing on GPU [41] [40] [39]. This can save us some trading time because we can not use the model to trade before it finishes the training.

1.2 Current State

Many interesting deep learning models are introduced in predicting the mid-price movement on Limit order book data [8] [9] [10] [12] . Some of the researchers are fascinated with building deep learning with feature engineering. A fully connected deep neural network with kernel function is introduced to classify the mid-price movement direction [6]. Convolution neural network with kernel function is also introduced by the same researchers [13]. A balanced mid-price is introduced as a new indicator [17].

$$Midprice_{Balanced} = \frac{Pirce_{ask} \times Volume_{ask} + Pirce_{bid} \times Volume_{bid}}{Volume_{ask} + Volume_{bid}}$$

It is tested on multi-layer fully connected neural network, convolutional neural network (CNN) and long short term memory (LSTM) neural network. Some researchers prefer no feature engineering. Raw limit order book with a convolutional neural network only [11], with a long short term memory neural network only [14], and with CNN and LSTM combined [7] are well studied. All the works mentioned above are using the same limit order book dataset, which is introduced in a paper [15] as a benchmark dataset. The benchmark dataset contains many normalized handcrafted features [51] from limit order book data of five stocks at the London stock exchange. The five stocks are Kesko Oyj, Outokumpu Oyj, Sampo Oyj, Rautaruukki Oyj, and Wartsila Oyj. There are 10 days of limit order book data for those five stocks and with each day from 10:00 to 18:25. While those researchers are dedicated to making sophisticated new features and building complicated new models, they completely neglect the problems that come with the raw data.

However, the movement of mid-price does not give us enough information for short term trading. The most straightforward way to trade is to predict the bid-ask spread crossings [51] [56]. For example, at the time t, the best bid is \$10 and the best ask \$20. Then the mid-price is \$15. Assume we buy one share of the stock, which means we need to pay \$20. At the time $t+1$, the best bid becomes \$18 but the best ask is still \$20. Then the mid-price increases from \$4 to \$19. If we sell the stock now at \$18, we still lose \$2. If the best bid goes up to \$22, then the bid-ask spread crossing happens. Now if we sell the stock at \$22, we gain \$2. Thus, the most straightforward approach is to predict the bid-ask spread crossings. Then we know if we should go long, go short or do nothing.

There are a few attempts in predicting trading opportunities. Large scale distributed deep learning training system has been used to learn from terabytes of limit order book data [3] [4]. However, the experiment is still at analyzing the prediction accuracy. A profitable trading model was made by using a deep convolutional neural network [1] [5]. The trading experiment is done on Lloyds Bank, Barclays, Tesco, BT, and Vodafone stocks. The data is from on London Stock Exchange for the entire 2017 year. The best daily profit is less than 0.0004 GBP. The average daily profit is less than 0.0003 GBP. Unfortunately, the paper did not mention the initial investment. We use the average stock price as approximation of the initial investment. Then the best average daily return rate among the five stocks is 0.000121%.

1.3 Data Analysis and Preprocessing

The main goal of this thesis is to build a trading system by using deep learning. The data we use in the experiment is from https://lobsterdata.com. It contains one day limit order book data of five stocks from 9:30 to 16:00. The five stocks are Apple, Google, Amazon, Intel, and Microsoft. It contains both order book and message book while the benchmark dataset does not contain message book. The order book records the orders in the market whenever a trading action hits the market. The message book contains information of those actions(buy limit order, buy market order, sell limit order, sell market order, and cancel) and the time those actions happen. Before we feed the limit order book data into our model, we need to preprocess it. The limit order book is an unevenly spaced time series [4] [3]. This is because a trader can place a limit order at any time. The limit order book includes the time when each limit order takes place. We notice that some limit orders are placed at the same time. If we do feature engineering on this raw data directly, the speed of price change may happen to be infinity [51] [17] [2]. A common approach to preprocessing unevenly spaced time series is to transform it into an equally spaced time series by using linear interpolation. [52] [53] [54] [55]. In this way, we turn the limit order book into an equally spaced time series with *time interval* equals to 0.25 seconds. We also trade only at those moments. We will try to find profit $profit\ target_{train}$ opportunity within 100 *time interval*s. We choose the $profit\ target_{train}$ to be \$0.03. $time\ interval = 0.25$ seconds is short enough to tell that we make a profit not because of luck. Within 100 *time interval*s, there is about 20% of the time that we can make a profit from Apple stock. Therefore, our random guess will cause us to lose a lot of money. Thus, we can tell if we make money because of our algorithm. Furthermore, our classification model does not suffer an imbalanced data problem to predict doing nothing all the time [60] [61]. When the chance of making a profit is about 1% like Microsoft stock and Intel stock, we are highly likely to get a trivial model, which tells us to do nothing all the time.

Unlike many researchers who use several years limit order book data to train a complicated model, we only have one-day limit order book data. It contains sample data from 9:30 am to 4 pm on June 21th 2012. We use the data from 10:00 am to 1:00 pm for training and use the data from 1:00 pm to 4:00 pm for predicting and trading. With the limit amount of data, a simple model is our only choice. Universal approximation theorem states that neural networks can approximate any function, which makes feature engineering unnecessary. However, with a simple model, feature engineering becomes meaningful [30]. Many researchers feed the raw data to the convolutional layer. The raw data contains the price and volume. Calculating the linear combination of price and volume does not make sense. In this paper, we will treat the limit order book data as three matrices instead of one. We introduce matrix price, volume, and capital. Their sizes are all 100 by 10. The matrix price contains the highest 5 bid prices and the lowest 5 ask prices over 100 *time interval*s. We need to standardize data by subtracting the mean. The reason we choose subtracting the mean instead of z-score normalization is because it makes more profit. The matrix volume contains volume with respect to the price in matrix price. We apply a natural log to the matrix volume and standardize it by subtracting the mean. For the stock volume, the difference between 100 shares and 101 shares is not as significant as the difference between 1 share and 2 shares. Therefore the linear relationship of the number of shares does not satisfy the financial meaning. That's why we use the natural log to transform the data. Finally, the matrix capital is the element-wise product of the matrix price and matrix volume. It is generated inside the deep learning model. This matrix can measure the capital at each depth of the limit order book data.

In this thesis, we also dive deep into the analysis of the benchmark dataset and use our trading model to predict the mid-price movement. We point out that there is no information on time in the benchmark dataset. Therefore, all the experiments are done on an event base instead of time base. A potential problem will be multiple events happen at the same time. Assume we have 3 events happen at the same time: e_1, e_2 and e_3. If the event-based model tells us to trade at e_2, it is actually not doable in practice. Another problem in the benchmark dataset is the distribution shift. We do 9 tests on the benchmark dataset. The ith test is using the first i days' data for training and $i+1$th day's data for testing. The experiment is a 3-class classification task. The 3 classes are mid-price move up, mid-price move down and mid-pice without change. For test 1 to test 6, the distribution of those 3 classes is 2:2:1. For test 7 to test 9, the distribution of those 3 classes is 1:1:1. However, the distribution in the training data does not change much, which leads to performance drops in test 7, test, 8 and test 9.

1.4 Deep Learning Model

It this paper, we are going to build two classification models. One for predicting whether we should go long or do nothing. The other one is for predicting whether we should go short or do nothing. These two models are built by mainly convolution neural networks. Since we have two matrices as input and one matrix generated at the very first layer, we have three matrices to deal with. Therefore, we need to use three convolutional layers to handle those three input matrices. After the max-pooling layer and flatten layer, we will concatenate the output together. Then we will do a logistic regression to get a probability indicating how likely there is an opportunity to make a profit $profit\ target_{train}$ within 100 $time\ interval$s. We will train two independent models. One for long stock only and the other one for short stock only. The label for the long stock only model is going long or do nothing. The label for the short stock only model is going short or do nothing. These two models will be trained independently. We will also use these two models to predict and trade independently. Those two models will be trained and tested on the limit order book data from LobsterData.com.

In order to make a profit, we want to make a trade when we are quite confident that there is an opportunity to make a profit. Therefore we set a lower profit target in trading: $profit\ target_{trade}$\$0.01. We also set a high probability threshold: *probability threshold* $= 0.7$. When the model predicts a probability higher than 0.7 we will go long(or short) and once we can make \$0.01, we close the position immediately. This is extremely easy to set up by placing a limit order. In the end, we sum up the profit from the two models as the total profit. There are a lot of randomnesses in the model: random initialization, max pool layers and dropout layers. Therefore, we conduct the experiment 100 times on Apple stock to evaluate the average performance. We enter the market with \$600. The average total profit is \$0.02. The maximum total profit is \$0.13. The minimum total profit is \$0. The total profit is larger than \$0 during 70% of the time. The training is done on a server with a 6 Core i7-3770 CPU (no GPU). Each model takes 12 seconds to finish the training, which is negligible compared with our 3 hour trading period. Thus we get a profitable, efficient and consistent trading system.

we also introduce two models for a mid-price movement experiment: a regular mid-price movement model and a benchmark mid-price movement model. The regular mid-price movement model is almost the same as our trading model except it produces three number instead of one. We need that for 3-classes classification. The regular mid-price movement model will be trained and tested on the limit order book data from LobsterData.com. The benchmark mid-price movement model is deeper than the regular mid-price movement model. It has 12 convolution layers while the regular mid-price movement model only has 3. The benchmark mid-price movement model is designed for the mid-price movement experiment on the benchmark dataset. It is inspired by other papers [11] [7] about the mid-price movement experiment on the benchmark dataset.

1.5 Contributions of This Work

There are three major contributions to the limit order book trading research in this thesis. First, we improve the average daily return rate by more than 1000% comparing with the most current paper [5] on limit order book trading. The best daily return rate from the paper is 0.00012%. The name of the stock is Tesco with the average price of £190 and the profit is £0.00023. Our average return rate on Apple stock is 0.0033%. Our initial investment is \$600 and the average profit is \$0.02. This is a huge improvement of the trading task on the limit order book and it can be a benchmark on the limit order book from Lobsterdata.com. So far our average daily return rate is still lower than the risk-free interest rate. We use the treasury bill to calculate the risk-free interest rate. Therefore, our research is still at the theoretical level and it is not ready for industry usage. However, our deep learning model is trained on half-day limit order book data. We may be able to build a more sophisticated model to achieve better performance. Nevertheless, our deep learning model is a big step forward on the limit order book trading with deep learning.

Second, we apply the popular mid-price movement prediction task on the limit order book data from Lobsterdata.com. Our deep learning could be used as a baseline model even though the performance is not very good. The difference between our experiment and the experiment on the benchmark dataset is that we use a time base instead of an event base. Even though the benchmark dataset contains many handcrafted features [51], it still limits researchers' creativity. Our model will be a new start of the research on the limit order book data from Lobsterdata.com.

Third, we dive deep into the data analysis which is ignored by many papers of limit order book we introduce in this thesis. We uncover problems of distribution shift, Z-score normalization and label shift. The distribution shift can easily happen in stock data. Since financial markets are changing all the time, the information we extract from the old data may not be useful for the new data. This is a big problem in machine learning. During the training process, we are optimizing the model for the data which we do not want. It could lead us to a performance drop or a wrong model. Despite of this problem, our benchmark mid-price movement model beats the experiments on the benchmark dataset on f1 score before and after the distribution shifts. We also find problems about the Z-score normalization in the benchmark dataset. It involves using future information for data preprocessing and it can also produce wrong mid-price movement labels. This mistake happens in many papers [11] [7] [15] [17] [9] [14] and their extraordinary results cannot be trusted. The Z-score normalization on the entire dataset can make each feature at the same scale and centers at 0. This can help the gradient descent a lot. Therefore, it often leads to good model fitting and good performances [82]. The Z-score normalization is a good method but using the future data to do the normalization is incorrect.

CHAPTER 2

MACHINE LEARNING

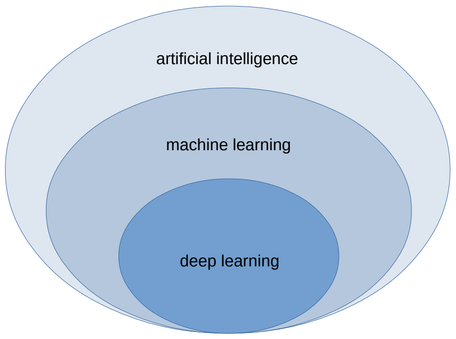


Figure 2.1: This is the relationship between artificial intelligence, machine learning and deep learning. Artificial intelligence does not have to use machine learning. For example, the chess AI Deep Blue. Deep learning is a part of machine learning. Logistic regression can be considered as a one layer deep learning model. Some deep learning models have thousands of layers. Deep learning is special because it can achieve better performance than other machine learning models on large dataset.

The figure 2.1 shows the relationship of artificial intelligence, machine learning, and deep learning. Artificial intelligence refers to machines, which is as smart as humans. It does not have to involve machine learning. For example, the famous chess AI Deep Blue was developed at 1980s. Is uses search algorithms to find the best move instead of machine learning. Those search algorithms are built entirely by humans. Researchers need to understand the logic behind it and provide the best end game solutions to Deep Blue. Even though Deep Blue was considered pretty good at that time, it can still lose to human players and it does not react very fast.

Machine learning is part of artificial intelligence, but it is quite different from the searching algorithm. We do not need to understand the logic behind the task, but we need datasets related to the task. If we want a machine to learn how to tell the difference between an apple and an orange, we just need to show it a lot of examples of apples and oranges. Then the machine will tell the differences between an apple and an orange. Mathematically, we need to chose the features we think are important and a model that can map the input features to the target. Then the machine will choose the best parameters for the model through the training. The training is an optimization process. The machine will keep updating the parameters in the model until the error between the output and the target is small enough. There are a lot of models we can choose from. For example, SVM, logistic regression, linear regression, decision tree and neural network are all machine learning models.

Deep learning is a small part of machine learning. A deep learning model is a neural network model with many hidden layers. Namely, the input is mapped by many functions until it becomes the output. A deep learning model can be much more complex than other machine learning models we just mentioned. There are many more parameters need to be determined than other machine learning models as mentioned. Thus, a deep learning model usually needs much more data to learn from. The performance of deep learning models can be very good. For example, Alpha Go beats the best human players in Go game. Since it needs a large amount of data, it takes the researchers a few weeks to train.

Thus, before we choose a machine learning model, we need to consider the data and time we have.

Before we start to talk more about deep learning, we will talk about some basic concepts in machine learning and how it is related to deep learning.

2.1 Logistic Regression

Logistic regression is one of the most widely used machine learning models in classification tasks [74] [75]. It is simple, fast and easy to interpret.

The formula of logistic regression is as follows.

$$y = \frac{1}{1 + e^{-(wx+b)}}$$

where x is the input vector, y is the output scalar, w is the coefficient vector and b is the bias scalar.

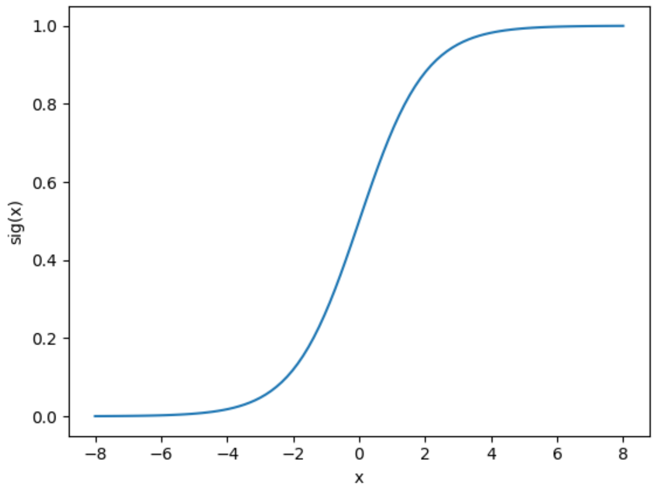


Figure 2.2: Logistic function is a function that maps $(-\infty, \infty)$ to $(0, 1)$. The output is usually interpreted as probability. With a probability threshold, we can classify the output by whether it is larger than the threshold or not. It is widely used as the final output in binary classification deep learning model.

Logistic regression can make a binary classification. There are two classes: class 1 and class 2. The output y is in (0, 1). There is a threshold h in [0, 1]. If $y \geq h$, then the logistic regression predicts class 1. Otherwise, the logistic regression predicts class 2. The common choice for h is 0.5 [76].

In statistics, the logit function [77] is defined as

$$logit(p) = ln(\frac{p}{1-p})$$

where p is a probability. $p \in (0, 1)$. The range of this function is $(-\infty, +\infty)$. This is a strict monotonically increasing function. $logit(p)$ goes to $+\infty$ when p gets closer to 1. $logit(p)$ goes to $-\infty$ when p gets closer to 0.

Let $logit(p) = wx + b$, then we get

$$\begin{aligned}
logit(p) &= wx + b \\
ln(\frac{p}{1-p}) &= wx + b \\
\frac{p}{1-p} &= e^{wx+b} \\
p &= (1-p)e^{wx+b} \\
p &= e^{wx+b} - pe^{wx+b} \\
(1 + e^{wx+b})p &= e^{wx+b} \\
p &= \frac{e^{wx+b}}{1+e^{wx+b}} \\
p &= \frac{1}{1+e^{-(wx+b)}}
\end{aligned} \tag{2.1}$$

Now we get the logistic function. Therefore, we interpret the output of the logistic function as a probability.

2.2 Cross Entropy Error Function

The cross entropy error function is a popular choice in classification tasks in machine learning [77]. The formula for discrete cross entropy error function is

$$E(x) = -\sum_{i=1}^{n} p(x_i)ln(q(x_i))$$

where $p(x_i)$ is the target probability of x_i from the data and $q(x_i)$ is the predicted probability of x_i from the machine learning model. Also $\sum_{i=1}^{n} p(x_i) = 1$ and $\sum_{i=1}^{n} q(x_i) = 1$. In classification problems, one of the $p(x_i)$ equals to 1 while the others equal to 0. $p(x_i) = 1$ means the data x belongs to class i. When $n = 2$, $q(x_i)$ is generated by sigmoid function. When $n > 2$, $q(x_i)$ is generated by softmax function.

2.2.1 Binary classification

The logistic regression uses the binary cross entropy as the error function. Here $n = 2$. We know $(p(x_1), p(x_2))$ can either be (0, 1) or (1, 0). Therefore, let $t \in 0, 1$, $p(x_1) = t$ and $p(x_2) = 1-t$.

The the error function becomes

$$E(x) = -(tln(\frac{1}{1+e^{-(wx+b)}}) + (1-t)ln(\frac{e^{-(wx+b)}}{1+e^{-(wx+b)}}))$$

.

Since one of t and $1-t$ will be zero, we assume $t = 1$ as an example. Then when we optimize the error function, we only need to optimize $-ln(\frac{1}{1+e^{-(wx+b)}})$ by gradient descent. We will need to increase $\frac{1}{1+e^{-(wx+b)}}$. Therefore, $\frac{e^{-(wx+b)}}{1+e^{-(wx+b)}}$ will decrease. Eventually, we may get $\frac{1}{1+e^{-(wx+b)}} = 0.85$. Since $0.85 \geq 0.5$, we predict x belongs to class 1 as expected.

2.2.2 Multi class classification

For example, in a three class classification $n = 3$. Then, there are three cases for $(p(x_1), p(x_2), p(x_3))$. It can be $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$. Assume it is $(1, 0, 0)$. Then the cross entropy error function becomes:

$$E(x) = -(1 \times ln(q(x_1)) + 0 \times ln(q(x_2)) + 0 \times ln(q(x_3))) = -ln(q(x_1))$$

When we train the model, the model will try to reduce $E(x)$ by gradient descent. Therefore, it will increase $q(x_1)$. Here

$$q(x_1) = \frac{e^{x_1}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

Since $q(x_1) + q(x_2) + q(x_3) = 1$, in order to increase $q(x_1)$, $q(x_2)$ and $q(x_3)$ will decrease. Eventually the $(q(x_1), q(x_2), q(x_3))$ will get close to $(1, 0, 0)$ but not equal to $(1, 0, 0)$. That is because $q(x_1)$,$q(x_2)$ and $q(x_3)$ are always larger than 0. Therefore the $(q(x_1), q(x_2), q(x_3))$ may become $(0.8, 0.1, 0.1)$. In terms of classification, we choose i with the largest $q(x_i)$ to be the predicted class. In our example, $(0.8, 0.1, 0.1)$ will be regarded as class 1 since 0.8 is the largest.

2.2.3 Softmax function

Logistic regression only works on binary classification. When it comes to multi-class classification, a common choice is softmax function [78]. Assume the input x is a n dimensional vector.

$$softmax(x) = (\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}})$$

There are two nice properties of softmax function. First, the sum of elements in $softmax(x)$ equals to 1. Second, if $x_i > x_j$, then $\frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} > \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$.

If x_i is the largest among all the elements in vector x, then $\frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$ is the largest among all the elements in vector $softmax(x)$. Then the input x will be classified as class i.

2.3 Training of Logistic Regression

Once we have the error function and the data, we can train our model. We use the binary classification error function as an example.

$$Error = -\frac{1}{n}\sum_{i=1}^{n}(y_i ln(\frac{1}{1+e^{-(wx_i+b)}}) + (1-y_i)ln(\frac{e^{-(wx_i+b)}}{1+e^{-(wx_i+b)}}))$$

.

The parameters w and b will be learned by gradient descent. We can initialize w and b with random numbers. Then we update w and b until the error is small enough. The purpose of the training process is to help the output get close to the target from the training data.

```
while $Error > \epsilon$ do
    $w = w - \lambda \times \frac{\partial Error}{\partial w}$
    $b = b - \lambda \times \frac{\partial Error}{\partial b}$
end while
```

ϵ and λ are constant. The $Error$ is recalculated every iteration because w and b will change. The $\frac{\partial Error}{\partial w}$ and $\frac{\partial Error}{\partial b}$ are calculated as follows:

$$\frac{\partial Error}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}(-\frac{y_i}{1+e^{-(wx_i+b)}} + \frac{1-y_i}{e^{-(wx_i+b)}(1+e^{-(wx_i+b)})})x_i$$

$$\frac{\partial Error}{\partial b} = -\frac{1}{n}\sum_{i=1}^{n}(-\frac{y_i}{1+e^{-(wx_i+b)}} + \frac{1-y_i}{e^{-(wx_i+b)}(1+e^{-(wx_i+b)})})$$

After that, the model will be ready to make predictions.

2.4 Logistic Function in Deep Learning

The logistic function is also called the sigmoid function, which is widely used in deep learning as an activation function [58]. It is also used as the final output function for classification tasks.

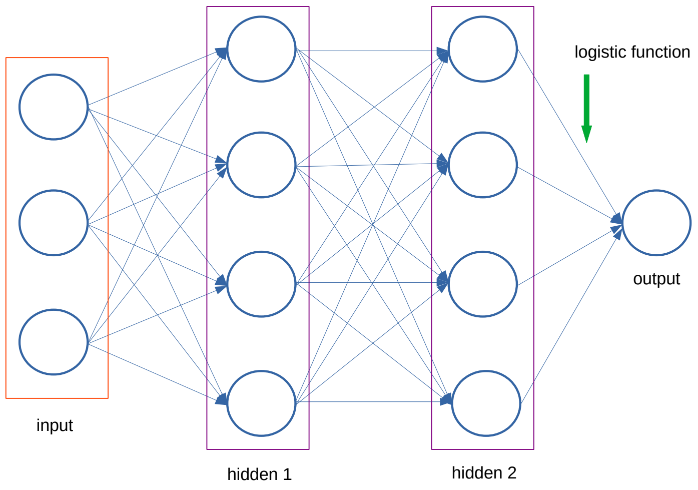


Figure 2.3: This is an example of deep learning model for binary classification. The arrows between the layers represent layers, which is the mapping function. Each column of circles represents the output matrix of each layer except the input. Each circle represents each element in the matrix. The arrows connecting all the elements of the matrix mean the mapping functions use all the elements in the matrix. In this example, the mapping function is matrix multiplication. For example, the matrix of hidden layer 2, which is 4 by 1, is multiplied on left by a 1 by 4 matrix to get a scalar. Then the logistic function is applied on the scalar to get the output. $output = \frac{1}{1+e^{-(w \times h_2 + b)}}$, where h_2 is the output matrix of the hidden layer 2. w and b are coefficients. w is the 1 by 4 matrix and b is a scalar. The error function of this model is the cross entropy function.

For example, in figure 2.3, it is a simple neural network model with two hidden layers. Assuming this model is for a binary classification task, then the last layer is exactly the logistic regression. That is why we also interpret the output of the neural network model as a probability. If we need to do a multi-class classification, then the last layer will be softmax.

Assuming figure 2.3 indicates a binary class function, now we can focus on the output.

$$output = \frac{1}{1+e^{-(w \times h_2 + b)}}$$

where h_2 is the output matrix of the hidden layer 2. w and b are coefficients. h_2 is a 4 by 1 matrix. After multiplied on left by a 4 by 1 matrix w, it generates a scalar. This scalar is mapped to the output by the logistic function.

Assuming the data is (x_i, y_i), where $i = 1, 2, \ldots, n$. y_i's are either 0 or 1. Then the cross entropy function will be used as the error function for this model.

$$Error = -\frac{1}{n}\sum_{i=1}^{n}(y_i \times ln(output_i) + (1 - y_i) \times ln(1 - output_i))$$

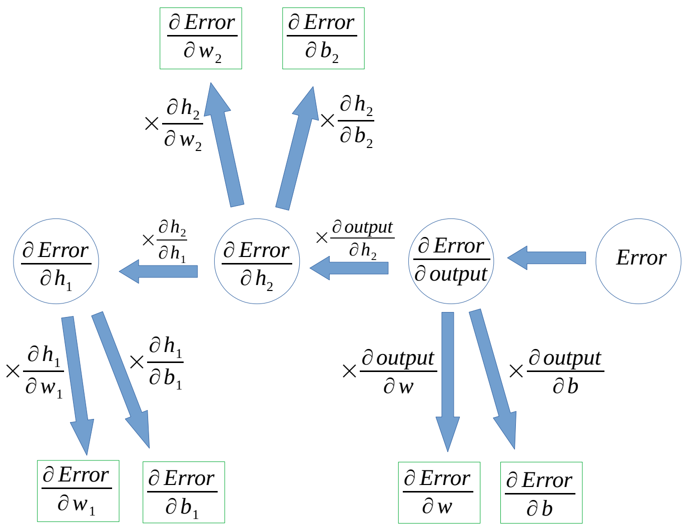
From last section, we show how to update w and b. However, there are other parameters we also need to update. For simplification, we assume

$$h_2 = w_2 \times h_1 + b_2$$

$$h_1 = w_1 \times input + b_1$$

2.4.1 Back propagation

If we want to update w_1, w_2, b_1, and b_2, we can calculate their gradient with respect to the $Error$ directly. We will use the chain rule to calculate the derivatives. This way, part of the calculation will be done more than once. We can save some intermediate results to simplify it.



$\frac{\partial Error}{\partial w_2}$
$\frac{\partial Error}{\partial b_2}$
$\times\frac{\partial h_2}{\partial w_2}$
$\times\frac{\partial h_2}{\partial b_2}$
$\frac{\partial Error}{\partial h_1}$
$\times\frac{\partial h_2}{\partial h_1}$
$\frac{\partial Error}{\partial h_2}$
$\times\frac{\partial output}{\partial h_2}$
$\frac{\partial Error}{\partial output}$
Error
$\times\frac{\partial h_1}{\partial w_1}$
$\times\frac{\partial h_1}{\partial b_1}$
$\times\frac{\partial output}{\partial w}$
$\times\frac{\partial output}{\partial b}$
$\frac{\partial Error}{\partial w_1}$
$\frac{\partial Error}{\partial b_1}$
$\frac{\partial Error}{\partial w}$
$\frac{\partial Error}{\partial b}$

Figure 2.4: Back propagation is a process of calculating derivatives by using the chain rule. If we consider the arrows as paths of calculating derivatives, we can see the path of $\frac{\partial Error}{\partial w_1}$ and $\frac{\partial Error}{\partial w_2}$ overlap. We can speed up the computation by storing intermediate results in the circle.

In figure 2.4 we can see that we can use $\frac{\partial Error}{\partial h_2}$ to calculate $\frac{\partial Error}{\partial w_2}$ instead of calculating $\frac{\partial Error}{\partial w_2}$ directly. In this way, we save a lot of computation steps. The process of this is the reversed order of calculating the output. Therefore, it is called the back propagation [63]. Here are the steps of back propagation for this example:

- calculate $\frac{\partial Error}{\partial output}$ and store it. Then $\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial output} \times \frac{\partial output}{\partial w}$ and $\frac{\partial Error}{\partial b} = \frac{\partial Error}{\partial output} \times \frac{\partial output}{\partial b}$.
- calculate $\frac{\partial Error}{\partial h_2} = \frac{\partial Error}{\partial output} \times \frac{\partial output}{\partial h_2}$ and store it. Then $\frac{\partial Error}{\partial w_2} = \frac{\partial Error}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$ and $\frac{\partial Error}{\partial b_2} = \frac{\partial Error}{\partial h_2} \times \frac{\partial h_2}{\partial b_2}$.
- calculate $\frac{\partial Error}{\partial h_1} = \frac{\partial Error}{\partial h_2} \times \frac{\partial h_2}{\partial h_1}$ and store it. Then $\frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial h_1} \times \frac{\partial h_1}{\partial w_1}$ and $\frac{\partial Error}{\partial b_1} = \frac{\partial Error}{\partial h_1} \times \frac{\partial h_1}{\partial b_1}$.

2.4.2 Training

Then the training process becomes:

```
while $Error > \epsilon$ do
    use back propagation to get $\frac{\partial Error}{\partial w}, \frac{\partial Error}{\partial w_1}, \frac{\partial Error}{\partial w_2}, \frac{\partial Error}{\partial b}, \frac{\partial Error}{\partial b_1}, \frac{\partial Error}{\partial b_2}$
    $w = w - \lambda \times \frac{\partial Error}{\partial w}$
    $b = b - \lambda \times \frac{\partial Error}{\partial b}$
    $w_1 = w_1 - \lambda \times \frac{\partial Error}{\partial w_1}$
    $b_1 = b_1 - \lambda \times \frac{\partial Error}{\partial b_1}$
    $w_2 = w_2 - \lambda \times \frac{\partial Error}{\partial w_2}$
    $b_2 = b_2 - \lambda \times \frac{\partial Error}{\partial b_2}$
end while
```

where λ and ϵ are constant. *Error* is recalculated for every iteration.

After that the model will be ready to make predictions.

2.4.3 Stochastic gradient descent

The formulas we show above are using gradient descent. The gradient descent uses all the data points to calculate the gradient. However, it will take a lot of time and computation resources to calculate the gradient by. It can cause memory overflow when we work with a large dataset. The solution is stochastic gradient descent. The stochastic gradient descent only use one data point at a time. The formulas look like this:

```
**while** $k <$ maximum number of iterations **do**
  **for** $i = 1, 2, \ldots, n$ **do**
    $Error = y_i \times ln(output_i) + (1 - y_i) \times ln(1 - output_i)$
    use back propagation to get $\frac{\partial Error}{\partial w}, \frac{\partial Error}{\partial w_1}, \frac{\partial Error}{\partial w_2}, \frac{\partial Error}{\partial b}, \frac{\partial Error}{\partial b_1}, \frac{\partial Error}{\partial b_2}$
    $w = w - \lambda \times \frac{\partial Error}{\partial w}$
    $b = b - \lambda \times \frac{\partial Error}{\partial b}$
    $w_1 = w_1 - \lambda \times \frac{\partial Error}{\partial w_1}$
    $b_1 = b_1 - \lambda \times \frac{\partial Error}{\partial b_1}$
    $w_2 = w_2 - \lambda \times \frac{\partial Error}{\partial w_2}$
    $b_2 = b_2 - \lambda \times \frac{\partial Error}{\partial b_2}$
    $k = k + 1$
  **end for**
**end while**
```

There is no randomness in the stochastic gradient descent.

2.5 Metrics

If we want to evaluate the performance of a machine learning model, we need metrics. The metrics are like scores that can help us determine how good a model is on a dataset. The metrics are the same for both training and testing. We want to introduce some metrics for binary classification. Before we talk about the metrics, we want to introduce some basic concepts first. True positive, false positive, true negative, and false negative are very fundamental concepts for binary classification. Assume we have a dataset for binary classification. The labels of the dataset are either positive or negative. Now we use the model to make predictions on this dataset. Then we have:

Table 2.1: True positive, false positive, true negative, and false negative

	real positive sample	real negative sample
predicted positive sample	True positive	False positive
predicted negative sample	False negative	True negative

The table 2.1 demonstrates the definition of true positive, false positive, true negative, and false negative. If a data sample is labeled positive and is classified as positive by the model, then it is a true positive (TP) sample. If a data sample is labeled negative and is classified as positive by the model, then it is a false positive (FP) sample. If a data sample is labeled positive and is classified as negative by the model, then it is a false negative (FN) sample. If a data sample is labeled negative and is classified as negative by the model, then it is a true negative (TN) sample.

Now we can introduce three metrics: precision, recall and f1 score.

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

$TP + FP$ are all the positive data samples the model predicts. The precision measures how many percentages of the positive data samples the model predicts have the real positive label.

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

$TP + FN$ are all the real positive data samples in the dataset. The recall measures how many percentages of the real positive data samples are found by the model.

$$f1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2\frac{precision \times recall}{precision + recall}$$

F1 score is the harmonic mean of the precision and recall. Precision and recall are both important for a model [84]. However, we cannot use both of them to choose the best model for us. For example, we have two models A and B. A has precision of 0.8 and recall of 0.6. B has precision of 0.7 and recall of 0.7. Then A has a better precision but B has a better recall. We cannot say A is clearly better than B or B is clearly better than A. If we only use f1 score, then there is no such a problem. We just need to choose the model with the best f1 score.

Suppose there are 100 data samples in the dataset, 60 of them are positive and 40 of them are negative. The prediction of the model is in the table 2.2 below.

Table 2.2: True positive, false positive, true negative, and false negative examples

	real positive sample	real negative sample
predicted positive sample	TP = 40	FP = 5
predicted negative sample	FN = 20	TN = 35

40 out of the 60 positive samples are classified as positive samples by the model. 20 out of the 60 positive samples are classified as negative samples by the model. 5 out of the 40 negative samples are classified as positive samples by the model. 35 out of the 40 negative samples are classified as negative samples by the model.

Therefore,

$$precision = \frac{TP}{TP + FP} = \frac{40}{40 + 5} = 0.889$$
$$recall = \frac{TP}{TP + FN} = \frac{40}{40 + 20} = 0.667$$
$$f1 = 2\frac{precision \times recall}{precision + recall} = 2\frac{0.889 \times 0.667}{0.889 + 0.667} = 0.762$$

Finally, we want to introduce a metric called Area Under The Curve (AUC). It is the area under the Receiver Operating Characteristics (ROC) curve. Before we define the ROC curve, we need to introduce true positive rate (TPR) and false positive rate (FPR).

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

We can see that TPR is the same as recall. Also we can see that TPR and FPR are in [0, 1].

Now we need to use the logistic regression as an example. The logistic regression is a binary classification model but its output is a number between 0 and 1. We need to pick a threshold to determine what number means positive and what number means negative. The default choice is

0.5. If the output is greater or equal to 0.5, it is classified as positive. Otherwise, it is classified as negative. When we change this threshold, we will get the ROC curve.

Assume the output of a logistic regression on a dataset are (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n). We also assume those output is sorted by y_i. y_1 is the smallest and y_n is the largest. The thresholds are those y_i's. The first threshold is y_1. Then if $y_i \geq y_1$, x_i is classified as a positive sample. Otherwise, x_i is classified as a negative sample. Since y_1 is the smallest, all the data samples are classified as positive. Therefore FN = TN = 0. Thus $TPR = \dfrac{TP}{TP+0} = 1$ and $FPR = \dfrac{FP}{FP+0} = 1$. Now we draw the first dot (1,1) on a FPR-TPR plane. The x-axis is the FPR and the y-axis is the TPR. The second threshold is y_2. Similarly, we calculate the TPR and FPR and draw the second dot (FPR, TPR) on the plane. Then we keep going until we use the last threshold y_n. Since y_n is the largest, all the data samples are classified as negative. Therefore, TP = FP = 0. Thus, the last point on the plane is (0, 0).

Now there are n points on the FPR-TPR plane. We use a straight line to interpolate those n points. The curve we get is the ROC curve. It is a curve goes from (1, 1) to (0, 0). The area under this curve is the AUC, so AUC is in [0, 1].

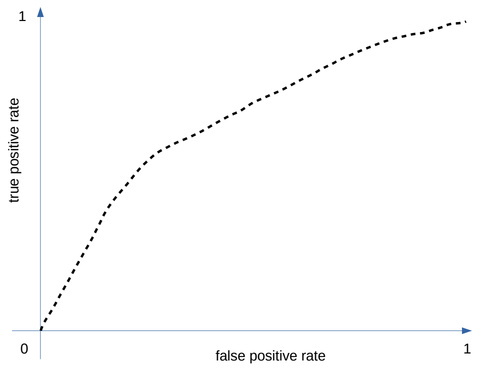


Figure 2.5: This is the figure of a ROC curve. The x-axis is the false positive rate. The y-axis is the true positive rate.

There is a special point on the FPR-TPR plane, which is (0, 1). That means FPR = 0 and TPR = 1. Namely, we classify every data sample correctly. There is no FP and FN. It represents a perfect model with a perfect threshold. That is the model we want to find. A good model should have a ROC curve close to (0, 1). Therefore, the area under the curve will be large. Thus, the AUC will be large. A large AUC means a good model [83].

So far we describe how logistic function is used in deep learning and why the output is considered as a probability. However, the network of the example deep learning model is very simple and we will not see such kind of deep learning model often. We will introduce some complex networks in the next chapter.

CHAPTER 3

DEEP LEARNING

3.1 Neural Network

Artificial neural networks are computing systems inspired by the biological neural networks in animal brains. The most basic unit in a neural network is a neuron. It takes the summation of the input and then applies an activation function to it. After that, the result will be passed to the next neuron.

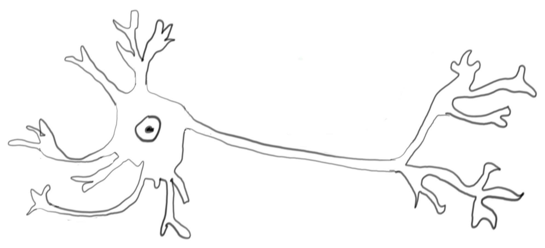


Figure 3.1: This is a figure of a human neuron.

Figure 3.2 is a mathematical demonstration of a neural network. It contains three layers: an input layer, a hidden layer, and an output layer. Neural networks could have more than one hidden layer but this one only has one hidden layer.

On figure 3.2, the input layer, the hidden layer, and the output layer are just vectors. The input x is a 3 by 1 vector. The hidden state h is a 4 by 1 vector. The output y is a 2 by 1 vector. We can see there are some arrows between those layers. Those arrows represent mappings between the vectors. The mathematical formula is

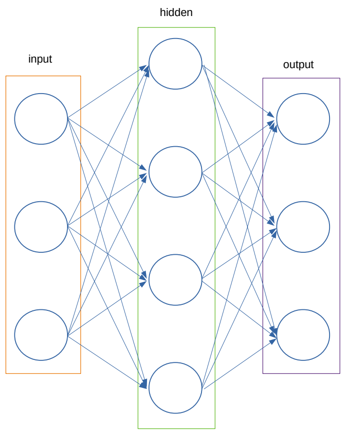


Figure 3.2: This is a very simple neural network. The arrows between the columns of circles represent layers, which are the mapping functions. Each column of circles represents the output matrix of each layer except the input. Each circle represents each element in the matrix. The arrows connecting all the elements of the matrix means the mapping functions using all the elements in the matrix. In this example, the mapping function is a matrix multiplication.

$$h = f_1(w_1 x + b_1)$$

$$y = f_2(w_2 h + b_2)$$

where w_1 is a 4 by 3 matrix, b_1 is a 4 by 1 vector, w_2 is a 2 by 4 matrix, b_2 is a 2 by 1 vector and $f_!$ and f_2 can be any activation functions. An activation function is a differentiable, nonlinear function [57]. The input of an activation function is actually a scalar but we pass a vector to it. That is because the activation function will be applied to every element of the input vector to generate the output. There are a lot of activation functions we can choose from. The purpose of those activation functions is to give neural network non-linearity because the reset operations in the neural network are all linear.

The definition of the activation function is

Definition 1. *f is an activation function if and only if f is bounded and $lim_{x->+\infty} f(x) = a$, $lim_{x->+\infty} f(x) = b$, $a \neq b$ [59].*

The definition implies the activation function needs to be a non linear function because the linear function is not bounded. Even though we need to calculate the derivative of the activation function during the training, the definition does not require the activation function to be differentiable. In fact, there are some activation functions that are not differentiable at some points in their domain.

Some commonly used activation functions are on the table as follows:

ReLU function

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & , x < 0 \\ 1 & , x \geq 0 \end{cases}$$

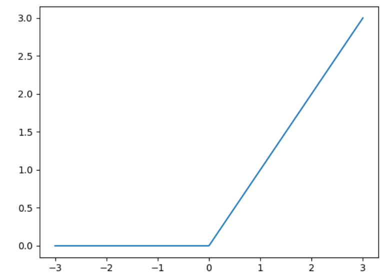


Figure 3.3: ReLU function

Sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

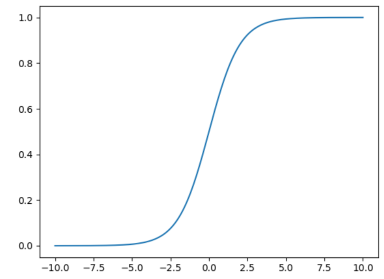


Figure 3.4: Sigmoid function

TanH function

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$f'(x) = 1 - f(x)^2$$

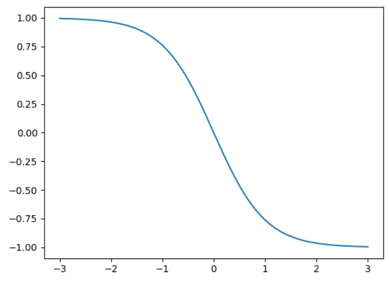


Figure 3.5: TanH function

ArcTan function

$$f(x) = tan^{-1}(x)$$

$$f'(x) = \frac{1}{1+x^2}$$

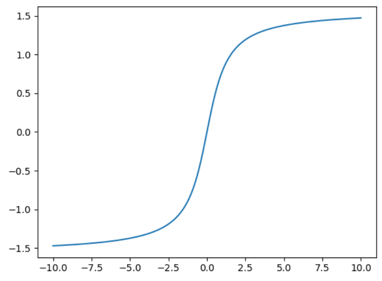


Figure 3.6: ArcTan function

Sinc function

$$f(x) = \begin{cases} 1, & x = 0 \\ \frac{sin(x)}{x}, & x \neq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & ,x = 0 \\ \frac{cos(x)}{x} - \frac{sin(x)}{x^2} & ,x \neq 0 \end{cases}$$

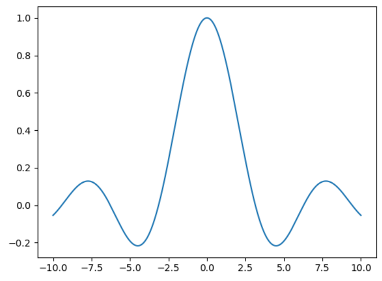


Figure 3.7: Sinc function

Binary step function

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

$$f'(x) = 0$$

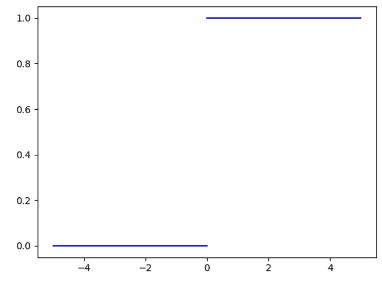


Figure 3.8: Binary step function

Mathematically the ReLU function is not differentiable at $x = 0$. The binary step function is also not differentiable at $x = 0$. Even though they are not strictly differentiable, we just need them to have a derivative numerically. We can define a value for the derivative at $x = 0$. Then we can treat them as differentiable functions and do all the calculations we need. We use $f'(0) = 0$ for both functions.

3.2 Deep Neural Network

The neural network model we mentioned so far is called a simple neural network because there is only one hidden layer. The figure below represents the differences between a simple neural network and a deep neural network. Neural networks with many hidden layers are called deep neural networks [58]. Deep learning refers to extracting useful information from data by using deep neural networks.

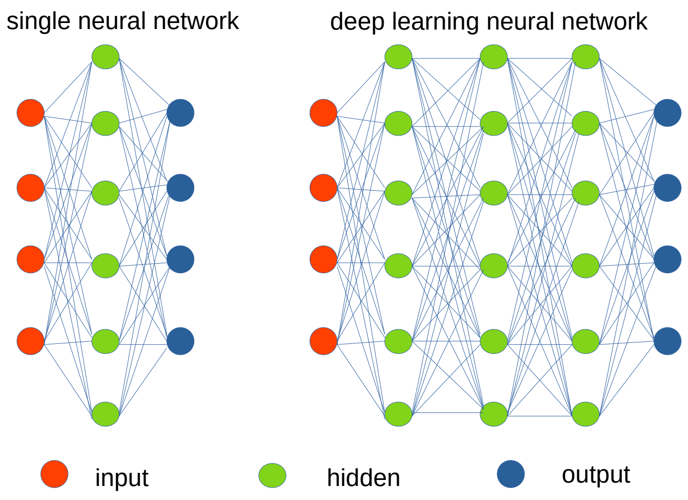


Figure 3.9: There are two deep neural networks. The one on the left has only one hidden layer. The one on the right has four hidden layers. The arrows between the columns of circles represent layers, which are the mapping functions. Each column of circles represents the output matrix of each layer except the input. Each circle represents each element in the matrix. The arrows connecting all the elements of the matrix means the mapping functions using all the elements in the matrix. In this example, the mapping function is a matrix multiplication.

As the following theorem states, a simple neural network can approximate any continuous function.

Theorem 3.2.1. (**universal approximation theorem**) [59]: Let $\phi(.)$ be an arbitrary activation function. Let $X \subseteq R^m$ and X is compact. The space of continuous functions on X is denoted by $C(X)$. Then $\forall f \in C(X)$, $\forall \epsilon > 0$: $\exists n \in N$, $a_{ij}, b_i, w_i \in R, i \in \{1...n\}, j \in \{1...m\}$:

$$(A_n f)(x_1, ..., x_m) = \sum_{i=1}^{n} w_i \phi(\sum_{j=1}^{m} a_{ij} x_j + b_i)$$

as an approximation of the function $f(.)$; that is

$$||f - A_n f|| < \epsilon$$

The norm in the last inequality can be infinity norm or p norm with $p \geq 1$.

However, the universal approximation theorem does not say how many neurons are needed in order to approximate a continuous function. In reality, if the target function is complicated, a lot of neurons are needed. Sometimes the number of neurons could be very large, which makes computation impossible. Therefore, many complicated neural works are used in practice. For example, convolution neural networks and recurrent neural networks.

3.3 Convolution Layer

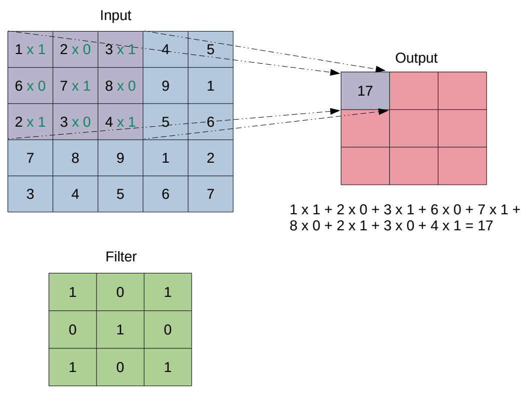


Figure 3.10: This is one step of a convolution operation. The input is a 5 by 5 matrix. The filter here is a 3 by 3 matrix. The output is 3 by 3. We use the filter to cover a 3 by 3 submatrix on top left corner of the input. Then we do an element-wised product of the filter and the submatrix. After adding up the product, we get the top left element of the output matrix.

We are going to focus on 2-dimensional convolution layer. The input of the 2-dimensional convolution layer is a 2-dimensional matrix. Assume its size is m by n. Now we need to define filter and stride. A filter is a matrix. The size of the filter is p by q. $p \leq m$ and $q \leq n$. The parameters in the filter will be trained during the training. The common choice for (p, q) are (2, 2), (3, 3) and (5, 5). The stride is a pair of scalars defined as (s_1, s_2). The common choice is (1,1).

The convolution operation is defined as below:

$$output_{i,j} = A(\sum_{s=0}^{p-1}\sum_{t=0}^{q-1} input_{i\times s_1+s, j\times s_2+t} \times filter_{s,t})$$

where $0 \leq i < \left\lceil \frac{m-p+1}{s_1} \right\rceil$ and $0 \leq j < \left\lceil \frac{n-q+1}{s_2} \right\rceil$.

$output_{i,j}$ indicates the output matrix of this convolution layer. The size of the output is $\left\lceil \frac{m-p+1}{s_1} \right\rceil$ by $\left\lceil \frac{n-q+1}{s_2} \right\rceil$. $A()$ can be any activation function. We can use $\circ$ as the convolution operation.

$$output_{i,j} = A((input \circ filter)_{(s_1,s_2)})$$

The (s_1, s_2) is the stride.

The advantage of the convolution layer compared to a simple fully connected layer is that the matrix filter is shared during the convolution operation. This dramatically reduces the number of parameters. Therefore it is less likely to overfit than a simple fully connected layer. We can use more than one filter, then the output will be a 3 dimensional matrix. We get the 3-dimensional matrix by stacking the $output_{i,j}$ of each filter.

The convolution layer has been widely used in computer vision [64] [65]. The convolution operation can deal with information in a local area on an image. The local area will be a matrix that has the same size as the filter. One of the common use case is edge detection [66] [67]. For example, the color of sclera and cornea are usually different. We can use this type of information to help to locate the eyes in people's faces.

Two famous examples of edge detection filters are using Prewitt operator [68] and Sobel operator [69]:

$$P = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Consider the following three matrix:

$$A = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix} \quad B = \begin{bmatrix} 10 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

as three matrices. We can consider matrix A only covers part of the sclera and matrix C only covers part of the cornea. Then matrix B covers part of the sclera and part of the cornea, which is the edge between the sclera and the cornea. Our input matrices A, B, and C have the same size as P and S. Those matrices are real examples of what happens in computer vision with convolution neural network. When the input matrix and the filter have the same size, there is only one submatrix we can use. Therefore we can ignore the stride. We use $\circ$ as the convolution operator for simplicity. We ignore the stride, because the input matrix and the filter have the same size. Then we get

$$\begin{aligned}
A \circ P &= \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 10 \times (-1) + 10 \times (-1) \\
&= 0 \\
B \circ P &= \begin{bmatrix} 10 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 10 \times 1 + 10 \times 1 + 10 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 0 \times (-1) + 0 \times (-1) \\
&= 30 \\
C \circ P &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 0 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 0 \times (-1) + 0 \times (-1) \\
&= 0
\end{aligned}$$

$$
\begin{aligned}
A \circ S &= \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 10 \times 1 + 10 \times 2 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 10 \times (-2) + 10 \times (-1) \\
&= 0 \\
B \circ S &= \begin{bmatrix} 10 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 10 \times 1 + 10 \times 2 + 10 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 0 \times (-2) + 0 \times (-1) \\
&= 40 \\
C \circ S &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \\
&= 0 \times 1 + 0 \times 2 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 0 \times (-2) + 0 \times (-1) \\
&= 0
\end{aligned}
$$

As you can see $A \circ P$, $C \circ P$, $A \circ S$ and $C \circ S$ are all equal to zero, which means the convolution filter does not find an edge. Actually, if all elements in matrix A are identical, then $A \circ P$ and $A \circ S$ are always zero. $B \circ P$ and $B \circ S$ are larger than zero which means there can be a boundary locally. The filter P and filter S above can detect edge vertically. They cannot detect a horizontal edge. It is also possible that there is not a boundary globally. Therefore, the convolution filter is only good at find information locally. For example, if the distance of useful elements in the matrix is larger than the size of the filter, then one convolution filter cannot catch them all.

One thing we need to remember is that we do not set and fix the elements in the filter. We get the elements in the filter by training the deep learning model, which means the training process will pick the right elements of the filter for us. We can initialize the filters with random numbers. Then use back propagation to calculate the gradient for each element. After that, we use stochastic gradient descent to update the elements in the filters until the error of the model is small enough. The process is similar to what we described at the end of last chapter.

In terms of finance, we can also interpret the filter as some kind of the derivative of the stock price. For example, $price_i$, $price_{i+1}$ and $price_{i+2}$ are three consecutive stock price. Assume $\Delta t = 1$. Then we know

$$
\begin{aligned}
price'_i &= \frac{price_{i+1} - price_i}{\Delta t} \\
&= price_{i+1} - price_i \\
&= price_i \times 1 + price_{i+1} \times (-1) + price_{i+2} \times 0 \\
&= \begin{bmatrix} price_i & price_{i+1} & price_{i+2} \end{bmatrix} \circ \begin{bmatrix} -1 & 1 & 0 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
price''_i &= \frac{\frac{price_{i+2} - price_{i+1}}{\Delta t} - \frac{price_{i+1} - price_i}{\Delta t}}{\Delta t} \\
&= price_{i+2} - 2price_{i+1} + price_i \\
&= price_i \times 1 + price_{i+1} \times (-2) + price_{i+2} \times 1 \\
&= \begin{bmatrix} price_i & price_{i+1} & price_{i+2} \end{bmatrix} \circ \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}
\end{aligned}
$$

Therefore, the filter can potentially generate features related to the derivatives of the stock price. However, this is based on the assumption that $price_i$, $price_{i+1}$ and $price_{i+2}$ are equally time spaced. Therefore, turning the limit order book into an evenly time spaced time series helps us interpret the model.

3.4 Max Pooling Layer

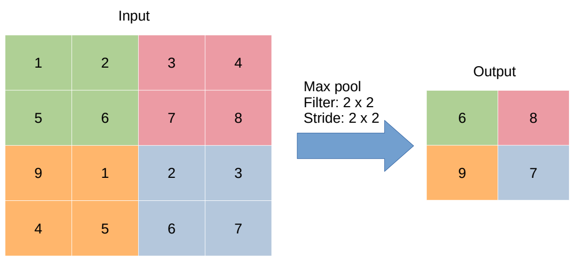


Figure 3.11: This is an example of 2-dimension max pooling. The filter size is 2 by 2. That means we need to pick the maximum element from a 2 by 2 submatrices. The stride in this example is (2, 2). Therefore the distance between the top left corner of adjacent submatrices is 2 (cells).

We are going to focusing on 2-dimension max pooling here. The input of the 2-dimension max pooling is a 2-dimension matrix. Assume its size is m by n. Now we need to define filter and stride. A filter is a matrix. The size of filter is p by q. $p \leq m$ and $q \leq n$. The common choice for (p, q) are (2, 2) and (3, 3). The stride is a pair of scalars defined as (s_1, s_2). The common choice is (1,1).

The max pooling operation is defined as below:

$$output_{i,j} = \max_{0 \leq s < p, 0 \leq t < q} input_{i \times s_1 + s, j \times s_2 + t}$$

where $0 \leq i < \left\lfloor \frac{m}{p} \right\rfloor$ and $0 \leq j < \left\lfloor \frac{n}{q} \right\rfloor$.

The matrix $output_{i,j}$ is the output of the max pooling layer. The size of it is $\left\lfloor \frac{m}{p} \right\rfloor$ by $\left\lfloor \frac{n}{q} \right\rfloor$.

The max pooling gets the largest element from each small region of the input matrix. It helps the model focus on certain elements and ignore some other less important elements. It can make the model more invariant.

The following figure is a comparison of a segment of a deep neural network with max pooling layer and a segment of a deep neural network without max pooling layer. The network on the left is the segment of a deep neural network with max pooling layer. The network on the right is the segment of a deep neural network without max pooling layer.

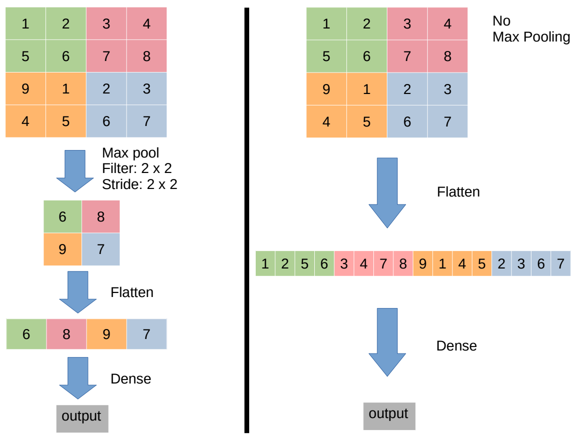


Figure 3.12: These are networks with and without maxpooling. The left figure is part of a neural network with max pooling. The right figure is part of a neural network without max pooling. The max pooling selects the largest element from submatrices. Therefore it reduces the size of the output matrix.

The last layer for both network is a simple fully connected linear layer.

$$y = f(wx + b)$$

where x is the input and y is the output. w is the weight and b is a bias term. $f()$ is an activation function. The major difference is that the x on the left is 4 by 1 while the x on the right is 16 by 1.

If only the 4 out of the 16 inputs are important, we do not need to consider the rest 12 inputs. Some people will argue that those 4 inputs can have large weights in the linear layer, then the network will ignore the rest 12 inputs. However, the position of the 4 inputs may change. For different input x, those 4 inputs could appear on different places of the original matrix. Therefore, we want to choose those important inputs and ignore the rest. Thus, by using max pooling, we securely assign zero as the weight to the rest 12 inputs despite their positions are not strictly fixed. This is why max pooling can make the neural network more invariant.

There is also one shortage of the max pooling layer. Since we ignore the rest 12 inputs, the weights in the previous layers which relate to those 12 inputs will not get any update. However, this fact is way less important than the invariant property in practice.

3.5 Dropout

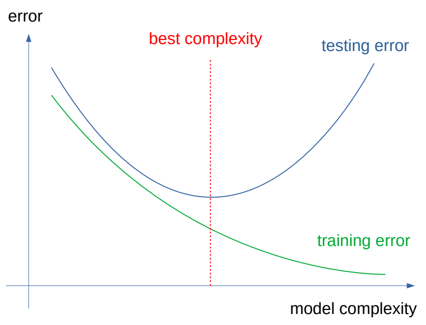


Figure 3.13: This figure demonstrates the change of training error and testing error with respect to model complexity. When the model is very simple, it cannot fit the data well. When the mode is complicated, it can fit the training data very well even if there are noise in the training data. Then the testing error will be much more than the training error.

Before we talk about the dropout layer, we need to address the overfitting problem in deep learning. The overfitting problem is a well-known problem in machine learning. It happens when the model is too complicated for the training dataset. It means there are too many degrees of freedom for the parameters in the model. Then the model gets a very small error on the training dataset while it gets a huge error on the testing dataset. One of the most important purposes of machine learning is to predict unseen dataset. This is extremely easy to happen with a deep learning model since it can get complicated easily. Even one linear layer can have hundreds of parameters. It is very hard to prevent a deep learning model from getting complicated.

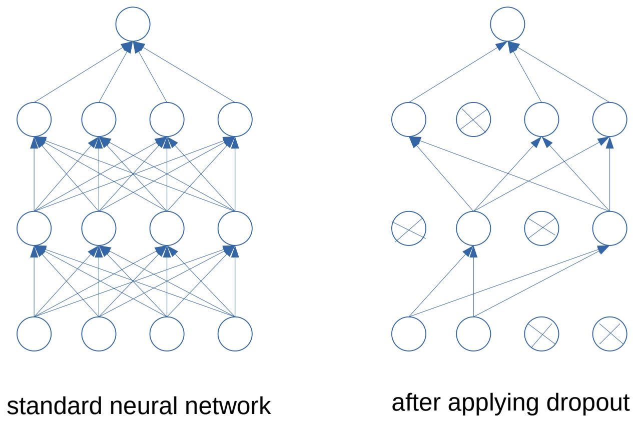


Figure 3.14: This is an example of how dropout works. The neural network on the left is a normal network. The neural network on the right is a network after being applied dropout. The arrows here show how the nodes from the previous layer are contributed to the nodes in the next layer. Before using dropout, the network is fully connected. This means all the nodes are used to calculate the nodes in the next layer. After using dropout, some of the nodes are set to be zero. Therefore those nodes are not contributed to the calculation of the nodes in the next layer.

Thus researchers discover many ways to prevent machine learning models to overfit. For example, the dropout layer is one of them. Dropout is a regularization technique. It is used to reduce overfitting in neural networks. When applying dropout to a layer, some of the nodes will be "invisible". They will be zero. It means those nodes will not contribute to the final output. Therefore, we do not need to do backpropagation on those nodes. If there are some parameters with a large gradient, those parameters will be much bigger than others. Then the model may rely heavily on those parameters. If those parameters are not the important ones, then the model may give us wrong indications. Therefore, making some parameters "invisible" will give parameters with a small gradient a better chance to be trained well. The "invisibles" are chosen randomly. We can set a dropout rate to decide how many percents of the nodes in the layer will be "invisible".

This random selection also contributes to the randomness of our deep learning model for trading. This means the output of this model can be slightly different each time we train the model.

3.6 Adam Optimization Algorithm

Adaptive Moment Estimation (Adam) algorithm is an optimization algorithm. It is widely used in deep learning training. Here is the formula.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \end{aligned}$$

θ_t is the parameter vector at step t. g_t is the gradient vector at step t. β_1 and β_2 are constant scalar. They are less than 1. A common choice for β_1 is 0.9 and a common choice for β_2 is 0.999. m_t and v_t are trying to measure a decaying average of g_t. m_t and v_t are initialized to be zeros. At the beginning of the training, m_t and v_t are close to zero. Therefore, we need to introduce $\hat{m}$ and $\hat{v}$ and use them to update the parameter θ_t.

We left out the $Error$ here. The $Error$ is calculated from an error function (ex. cross entropy function) as we mentioned in the last chapter. The $g_t = \frac{\partial Error}{\partial theta}$. The goal of the Adam optimization algorithm is to minimize $Error$.

Assume our deep learning model is $y = f(x, \theta)$. x is the input, y is the output and θ is all the parameters in the model. The g_t is the gradient of θ of one mini batch of the dataset. m_t and v_t are running averages.

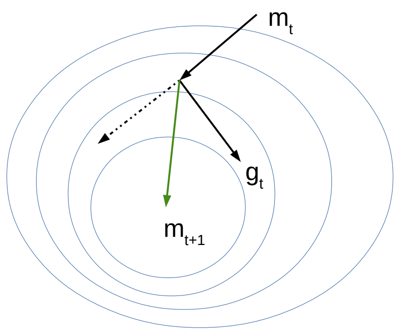


Figure 3.15: This is how momentum is updated at each step. g_t is the gradient of parameters at each step. m_t is the moving average of g_t with decay. $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$, where β_1 is a constant in (0, 1). m_t is the real direction of the gradient descent.

The point of have m_t and v_t as running averages is to prevent the zigzag derivative situation in the figure above. When we use a mini batch of the dataset to calculate the derivatives, it is possible to get g_t pointing to the wrong direction. By using a running average, we can accomplish our goal in the zigzag situation. In gradient descent, the change of a parameter is depended on the product of the gradient of the parameter and the learning rate. Assume there are only two parameters a_1 and a_2 in a model. Their initial values are both 0. Their optimal values are both 10. Assume the gradient of a_1 is much larger than the gradient of a_2. Then the path of (a_1, a_2) during the optimization can be: (100, 1), (-90, 2), (80, 3), (-70, 4), ..., (10, 10). a_1 has a large gradient, so it can change a lot before it converges. It goes to 100 first. Then it knows 100 is too much, so it goes the opposite direction. Again -90 is too much, so it goes the opposite direction. Their path is zigzag.

One way to fix this is to reduce the learning rate. We can make the learning rate very small. Therefore, the changes on the parameters will also be small. Then the path may become: (1, 0.01), (2, 0.02), (3, 0.03), (4, 0.04), ..., (10, 0.1), ... That would take too many steps until the second parameter goes to 10. We can see the learning rate can control how fast the parameters change. By reducing the learning rate, we fix the zigzag problem but we have another problem. It slows down the entire optimization process. Thus, reducing the learning rate is not a solution to the zigzag problem.

If we can scale the data and make the gradients of all the parameters at the same level, then there will not be zigzag problem. However, this is beyond the scope of an optimization algorithm. What an optimization algorithm can do is to use the running average to cancel out part of the gradients in the opposite direction. The momentum or running average can help the parameters focus on moving in one direction and the parameters can converge faster.

Another good point of the Adam optimization algorithm is that it can adjust the "learning rate". The learning rate η is constant but $\hat{v}_t$ can change. If we consider $\dfrac{\eta}{\sqrt{\hat{v}_t}+\epsilon}$ as the "learning rate", then it is fully controlled by $\hat{v}_t$. When $\hat{v}_t$ is small, it means the "learning rate" is large and the gradient is small. If we consider the dataset as a surface, then we are on a flatter surface. Therefore, we can move faster to minimize the loss function. This helps a lot at the end of the training. On the other hand, when $\hat{v}_t$ is large, we are on a steep surface. Therefore, we need to be more careful in case of running in the wrong direction. For example, we can run into some local minimum we don't want. We may eventually get out of that local minimum but it will definitely take more steps. Thereby, our training process will slow down.

Now we use a simple example to show how a convolution neural network is trained. Assuming the input and output pairs in the training data are $(x_i, \hat{y}_i)_{i=1}^n$, there is only one filter in the convolution layer. The matrix of filter is f. We use $\circ$ as the convolution operator. We ignore the activation functions for simplicity.

The model in figure 3.16 is defined as:

$$z_1 = x \circ f$$

$$z_2 = MaxPool(z_1)$$

$$z_3 = Flatten(z_2)$$

$$z_4 = Dropout(z_3)$$

$$y = w \times z_4 + b$$

The parameters in the model are f, w and b. There is no parameter in max pool layer, flatten layer or dropout layer needs training. z_4 is just part of z_1. After we get y, we need to pick an error function to calculate the error. For example, we use mean squared error function.

$$Error = \frac{1}{n}(y_i - \hat{y}_i)^2$$

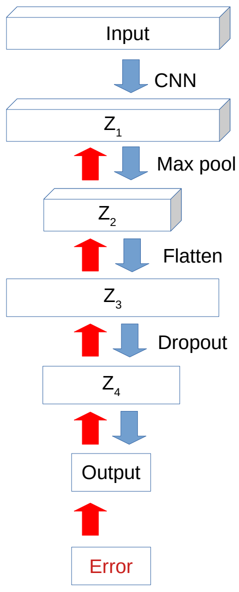


Figure 3.16: This is a simple convolution neural network. There is one convolution layer, one max pool layer, one flatten layer and one dropout layer. The blue arrows represent the mapping of the data from the input to the output. The red arrows represent the direction of back propagation.

Now we can use back propagation to calculate $\frac{Error}{f}$, $\frac{Error}{w}$, and $\frac{Error}{b}$. Then we can optimize f, w and b as what we did in last chapter by using gradient descent.

```
while $Error > \epsilon$ do
    use back propagation to get $\frac{Error}{f}$, $\frac{Error}{w}$, and $\frac{Error}{b}$
    $g_t^f = \frac{\partial Error}{\partial f}$
    $g_t^w = \frac{\partial Error}{\partial w}$
    $g_t^b = \frac{\partial Error}{\partial b}$
    use Adam algorithm to update $f$, $w$, and $b$.
end while
```

There is a lot of randomness in the model.

- Random initialization

  We need to initialize f, w and b with random numbers.

- Max pool
  The max pool layer picks the largest number from each submatrices. The position of the largest number with respect to each submatrix is not fixed.

- Dropout
  z_4 is randomly selected from z_3.

The random initialization has quite less effect in the training compared with max pool and dropout. Usually, we do not care too much about the starting point of an optimization process. The max pool and dropout have a large impact on the gradient calculation. The elements not selected by the max pool or dropout will have gradient equaling to zero in the back propagation process. It's not a problem because z_1, z_2, z_3 and z_4 are not parameters we need to train. However, when we calculate $\dfrac{\partial Error}{\partial f}$ by using $\dfrac{\partial Error}{\partial z_1}$, the results will be affected a lot.

There is no randomness in gradient descent at all. There is no randomness in stochastic gradient descent if the order of the training data is fixed.

CHAPTER 4

EXPERIMENT ON LIMIT ORDER BOOK FROM LOBSTERDATA

4.1 Actions in the Market

Nowadays, a lot of tradings are done in the New York Stock Exchange (NYSE), NASDAQ, and the London Stock Exchange (LSE). Those stock exchange centers use the limit order book system, so traders are making trades in such an environment. There are limited actions a trader can make in such trading systems.

- place market order
  - market bid order
    A market bid order can let a trader buy a certain stock at the current market price, which is the best ask price. However, if the volume of the best ask is less than what the trader wants, then this market order will let the trader buy the next best ask. This market order will stop until the trader buys the number of shares of stock he wants. Market orders usually get executed immediately.

    For example, a trader wants to buy 100 shares of stock S at market price. The current best ask price is \$90 and the volume is 80 shares. The second best ask price is \$91 and the volume is 200 shares. Then the trader will buy 80 shares at price \$90 and 20 shares at the price of \$91.
  - market ask order
    The market ask order is for a trader to sell a stock at the market price. The mechanism is similar to market bid order.

- place limit order
  - limit bid order A limit bid order is for a trader to buy a certain stock at a fixed price. Once a trader submits a limit bid order, the limit bid order will be placed in the system in the order of the bid price. The limit order with the lowest bid price will be executed first. Whenever another trader is willing to sell the same stock at the market price, the limit order will be executed immediately. If no one wants to sell it at that price, the limit order won't be executed. It will keep waiting.

For example, trader A wants to buy 80 shares of stock S for $100. Trader B wants to sell 30 shares of stock S for $100. Then only 30 shares of stock S are traded. Trader A's limit order becomes biding for 50 shares of stock S for $100.

  - limit ask order The limit ask is for a trader to sell a stock at a fixed price. The mechanism is similar to limit bid.

- cancel order

  If a trader changes his mind, he can cancel a limit order before it is executed. Since the market order usually executes immediately, it is unlikely a market order can be canceled.

4.2 Limit Order Book

In the trading experiment of this chapter, we use the limit order book of the official NASDAQ Historical TotalView-ITCH sample on 06/21/2012. We will also use it in the mid-price movement experiment at next chapter. It can be downloaded from https://lobsterdata.com. The sample files contain an order book file and a message book file. We use the depth 5 sample file as an example. Depth 5 means that there are the 5 lowest limit ask orders and the highest limit bid orders in each record. Whenever there is a change in the 5 lowest limit ask orders and the highest limit bid orders, there is a new record in the data. The depth 5 data contains 301587 rows of data. It starts at 9:30 am and ends at 4 pm. On average, an event occurs every 0.0776 seconds.

- order book

  In the order book, each row is a snapshot of current limit order book status. Here is one row of a depth 5 order book sample data: "5859400, 200, 5853300, 18, 5859800, 200, 5853000, 150, 5861000, 200, 5851000, 5, 5868900, 300, 5850100, 89, 5869500, 50, 5849700, 5".

  Let's consider the first 4 numbers. 5859400 means the lowest ask price is 585.94. And the volume is 200 shares. 5853300 means the highest bid price is 585.33. And the volume is 18 shares. For the second, those are the second lowest ask, ask volume, second highest bid and bid volume. Every four numbers of the row are the ith lowest ask, the ith ask volume, ith highest bid and the ith bid volume, for $1i \leq 5$. Our data sample is depth 5, which means it contains the best 5 bids and the best 5 asks. The best 5 bid prices are 585.94, 585.98, 586.1, 586.89, 586.95. And their volumes are 200, 200, 200, 300, 50 accordingly. The best 5 ask prices are 585.33, 585.3, 585.1, 585.01, 584.97. And their volumes are 18, 150, 5, 89, 5 accordingly. Figure 3.1 below gives us a better view of the market at the moment.

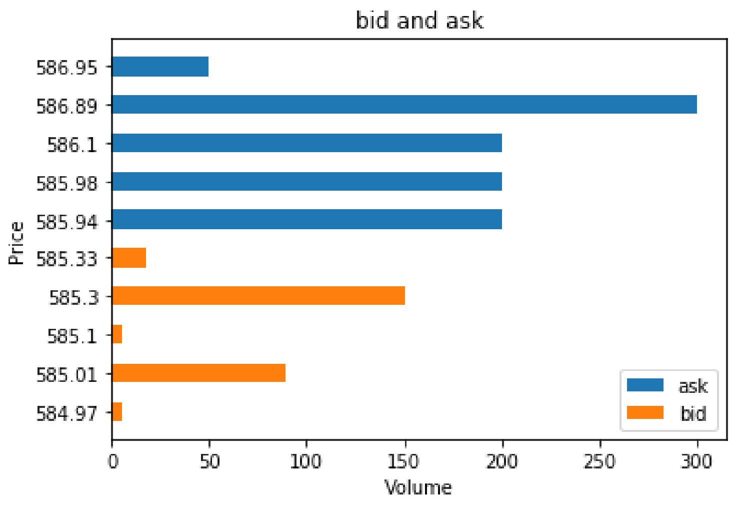


Figure 4.1: This is a snapshot of the limit order book at a certain moment. There are 5 ask orders and 5 bid orders. The x-axis is the number of shares. The y-axis is the price of the orders.

- message book

  An important fact of the order book data is that it is a time series. All those limit orders are recorded based on when it happens. However, there is no time information in the order book. The time information is in the message book. Whenever a trader submits a market order or a limit order, there is a record in the message book. We call this an event. Since a trader can submit a market order or a limit order at any time, the time difference between each event is not the same. For example, this is a segment of the message book data.

Table 4.1: A segment of the message book

34200.004241176	1	16113575	18	5853300	1
34200.00426064	1	16113584	18	5853200	1
34200.004447484	1	16113594	18	5853100	1
34200.025551909	1	16120456	18	5859100	-1
34200.025579546	1	16120480	18	5859200	-1
34200.025613151	1	16120503	18	5859300	-1
34200.201517942	1	16166035	100	5859300	-1
34200.201735987	3	16113594	18	5853100	1
34200.201742395	3	16113584	18	5853200	1

The first column is time and the unit is second. The second column is called type and there are 5 types:

- 1: Submission of a new limit order
- 2: Cancellation (Partial deletion of a limit order)
- 3: Deletion (Total deletion of a limit order)
- 4: Execution of a visible limit order
- 5: Execution of a hidden limit order

  A hidden limit order is a limit order that is not visible on the limit order book. It is still active and it is ranked the same as the visible limit order. The hidden order always gets traded before the visible order if their prices are the same. For example, If the price of the best bid limit order is $ 10 and the number of shares is 100, there is a hidden limit bid order with price equal to $ 10 and the number of shares is 30. If a trader places a market ask order for 50 shares, then the 30 shares of the hidden bid order are sold first. Then the 20 shares of the visible bid order are sold. The mid-price calculation does not use the hidden order.

The third column is the order id. The fourth column is the number of shares. The fifth column is Price: Dollar price times 10000 (i.e., A stock price of $91.14 is given by 911400). The Sixth column is Direction:

- -1: changes happen at sell limit order
- 1: changes happen at Buy limit order

We can see the time difference between row 2 and row 3 is about 0.0002 second and the time difference between row 4 and row 5 is about 0.00002, which is 10 times smaller. With the message book data, we can measure how fast the price changes. Many researchers like to model stock as a physical object [70] [71]. The data from the message book can tell how it moves and what the acceleration is and that information is not available on order book.

By looking into the message book, we notice that some records happen at the exact same time, even though the accuracy of a timestamp is nanosecond. It could be because those limit orders are submitted together by one trader. Therefore, the time difference between the records is 0. If we calculate the derivative of those prices, we will get infinity. Many papers fail to mention that before they do feature engineering or normalization [12] [7] [9] [15] [16] [17]. We will use the last record in the limit order book when there are multiple records show up at the same time. This is an indirect reason we want to turn the limit order book data into an evenly spaced time series.

4.3 Data Analysis and Data Preprocessing

The limit order data we have is actually an unevenly spaced time series. Before we apply machine learning to it, we want to turn it into an evenly spaced time series. This is not only a common approach in time series analysis [52] [53] [54] [55], but also a practical solution to our real time trading system.

There is a flaw in building real time trading system on event base algorithm. As we have mentioned, some events will show up at the same time. For example, there are 5 consecutive events: e_1, e_2, e_3, e_4 and e_5. Assume e_3, e_4 and e_5 happen at the same time. Now suppose we want to use the most recent 3 events to make predictions, then we should use e_1, e_2 and e_5 instead of e_1, e_2 and e_3 because e_5 is the final state of the market at that time. However, despite that e_3, e_4 and e_5 happen at the same time, the computer will receive them in order. Then our event base real time trading system will react when receiving e_3 and use e_1, e_2 and e_3 to make prediction and trade. We won't realize the mistake before we receive e_4 and e_5.

An evenly spaced time series may lose some information if multiple trading actions happen during one time interval. Even for the event base algorithm, we do not need multiple records in one nanosecond, which is a special case of the time interval that equals one nanosecond. The benefit of this evenly spaced time series is that the model no longer needs to consider time. Since it is an evenly spaced time series, only the order of the data matters. That is easy to deal with. We choose the length of the *time interval*s to be 0.25 seconds. One reason we choose the time interval to be 0.25 seconds is that it is easier to set up a limit order on a trading platform(like Quantopian [72]) on a time base than an event base. In our experiment, we predict future mid-price in 100 seconds. If we predict there is an opportunity in 100 *time interval*s, then we can submit a limit order that only lasts 100 *time interval*s to the trading platform. Then the trading platform can take care of the rest. Another reason is that we want to choose a time interval that can avoid the imbalanced data problem. We want the model to do a classification task. The output is a probability of whether we shall trade or do nothing. We will choose a *probability threshold.* If the probability is higher than the *probability threshold*, we trade. Otherwise, we do nothing. If we choose the *time interval* to be too short, it can be very hard to make a profit within 100 *time interval*s. If the probability is less than 0.001, then it is very hard for a machine learning model to learn. The output of the model may always be doing nothing. If we choose the *time interval* to be too long, then it will be easy to make a profit within 100 *time intervals*. It will be hard to judge whether we make money because of luck or not. The figure below shows the relationship between the *time interval* and the probability of making a profit.

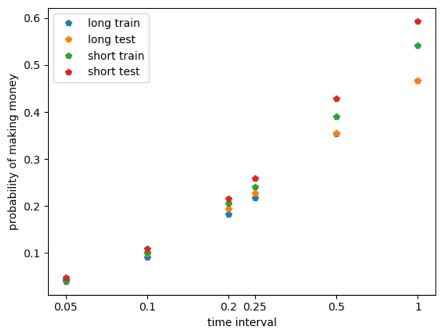


Figure 4.2: This is the relationship between the probability of making a profit and the *time interval*. The probability is calculated directly from training and testing dataset. The blue dot and the orange dot overlaps when *time interval* = 0.25 and 1. We use a natural log to scale the x axis.

The figure below is the predicted probability of making money with different *time interval*. The convolutional deep learning model we use will be introduced later. We choose the probability threshold to be 0.5.

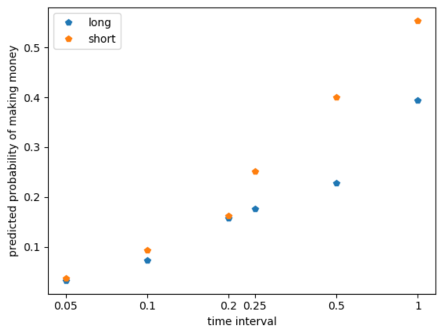


Figure 4.3: This is the relationship between the predicted probability of making a profit and the *time interval*. We first train our model on the training dataset. Then, predict the probability of making a profit on the testing dataset. The probability in this figure is the probability of the model predicted on the testing dataset. We use a natural log to scale the x axis

This shows that when *time interval* = 0.05, the predicted probability of making money is less than 0.04 for both the long model and the short model. That means more than 96% of the time the model will ask us to do nothing. If we make the *time interval* lower, it will lead us to the imbalanced data problem. For example, we have a binary classification task. The output is either 0 or 1. We have m 0s and n 1s in the training data. Assume we have a trivial model that the output is always 0 and the error function is the mean square. Then error will be $\frac{n}{m+n}$. If $m >> n$, then $\frac{n}{m+n}$ will be almost zero. Therefore, the trivial model will be considered as a very good model. If we train a machine learning model on this dataset with gradient descent, we can easily run into the trivial model we have just mentioned. Even though this trivial model is a local minimum, it is very hard for the gradient descent to get out of that local minimum. Thus, the output of the model will always be 0.

After we choose the *time interval* to be 0.25 seconds, the probability of making a profit by long stock only and short stock only are all between 0.2 and 0.3 based on the data. Therefore, the model does not suffer the imbalanced data problem much and it is hard for a simple trading strategy to make a profit.

Even though we do not have the data at every 0.25 seconds, we can still generate it. For example, we want the state of the market at time 34300, but we only have the record at 34299.999999999 and 34300.000000001 according to the message book. Then we know that the state of the market at 34300 is the same as the record at 34299.999999999. Because no trade happens until 34300.000000001. Therefore, for time $\in [34299.999999999, 34300.000000001)$, the data does not change. Therefore, we can use the data at 34299.999999999 as the data at 34300. This way we can generate the regular time series from the limit order data.

4.4 Design of the Trading System

We have two trading systems. One for long stock only and one for short stock only. Each system contains a deep learning model. These two systems will be executed in parallel in the real world. We will introduce the system for long stock only first. The data we use is the evenly spaced data. At every moment, we want to use the most recent 100 data points to predict if we can make a profit $profit\ target_{train}$(0.03 dollar) in the future 25 seconds(100 *time interval*s) by going long at the moment. Therefore, each input and target of the deep learning model is generated from 200 consecutive data points. We have the stock data for one day. We pick the middle ($MidT$) of that trading period. Then we can split those data evenly into two parts. The first part is the data before $MidT$ and the second part is the data after $MidT$. We will use the first part of the data to train the model and then use the second part of the data to do predicting. Finally, we do trading according to the prediction and calculating the profit. The trading actions are as follows:

If we find the model predicting the probability of making a profit is higher than the probability threshold, we will place a market bid order first to buy one share of the stock. Then we place a limit ask order at price $current\ best\ ask + profit\ target_{trade}$(0.01 dollar) for one share. This means we will sell the one share of stock whenever the bid price meets $current\ best\ ask + profit\ target_{trade}$. However, there is a chance that we cannot sell our stock within 25 seconds. Then we will cancel the limit order, and sell the stock at market price. If we find the model predicting the probability of making a profit lower than or equal to the probability threshold, we do nothing.

The system for short is quite similar to the system for long. The design of the deep learning model for long and short are the same. The input data is also the same. The only difference is the target. The target of the model for short is whether we can make a profit in the future 25 seconds if we go short at the moment. The trading action of the system for short is going short instead of going long.

If we find the model predicting the probability of making a profit higher than the probability threshold, we will short one share of the stock at market price. It means we will borrow one share of the stock and use the market ask order to sell it. And we place a limit bid order at price $current\ best\ bid - profit\ target_{trade}$ for one share. This means we will buy back the one share of stock whenever the ask price meets $current\ best\ bid - profit\ target_{trade}$. However, there is a chance that we cannot buy back our stock within 25 seconds. Then we will cancel the limit order, and buy the stock at market price. If we find the model predicts the probability of making a profit lower than or equal to the probability threshold, we do nothing.

So far we only consider how to make money. We have not considered how to control the loss yet. If we buy one share of a stock and then the stock price decreases a lot, we will lose money. Since we hold the stock for just a few seconds, the stock price will not change too much. We do not need to worry about losing a billion dollar. We can easily set up a loose bound for the profit to prevent us from big loss.

```
if position is opening then
    if profit $\geq$ $profit\ threshold$ or profit $\leq$ loose bound or $holding\ time$ reach the limit then
        close position
  end if
else
  if predicted probability $\geq$ $probability\ threshold$ then
        open position
  end if
end if
```

For example, bound = \$1, which is 100 times $profit\ target_{trade} = \$0.01$. The profit does not hit this bound within holding time limit (25seconds) for all five stocks (Apple, Google, Amazon, Intel, Microsoft). This is a fact from the dataset. Therefore, it has no effect on our trading model unless something extremely bad happens (ex: market crashes). If the market crashes, we will lose at most \$1.

However, a tight bound is a different story. For example, if the bound is small. Then it may close a position before we reach the profit target. That means we lose money before we reach the profit target. This is highly related to the variance of the stock. Different stocks need different tight bound. It is hard to set up a tight bound in general. Thus, we do not use the profit bound in this trading experiment.

For example, our algorithm lose \$0.32 on Amazon stock on average, which will be discussed in detail later. When we add a profit bound = \$0.1, we lose \$0.98 on average. When we add a profit bound = \$0.2, we lose \$0.36 on average. Therefore, this simple tight profit bound is not helpful.

4.5 Assumption

There are four things we need to mention. First, the stock exchange charges a fixed amount of commission fee per day from a high frequency trader. Some exchanges even free the commission fee. So we do not need to worry about how many times we trade a day. Second, we will hold or short at most one share at any time. The reason why we do this is that if we only trade a small number of shares it is unlikely we will influence the price of the stock. Therefore, the market order is almost guaranteed to be executed immediately. The only thing to worry about is that another high-frequency trader might be faster than us. Thus, we need to assume we are the fastest trader in the stock exchange center. Last, we need to assume we don't need to pay extra money to short a stock. Before a trader can short a stock, he needs to open a margin account and have enough money in it. He may also need to pay interest for the stock he borrowed. All that cost is up to the stock exchange. We will not consider those in our experiment. Last, we need to assume that the model training time is 0. Then we can use the model immediately after training.

4.6 Experiment Design

Before we start training we need to do data preprocessing. As we mentioned early, we can turn the limit order book data into a evenly spaced time series. We pick the time window to be 0.25 second, then the data becomes $x_t := x_1, x_2, \ldots, x_n$. We need the time information from the message book to determine which data record to select from the order book. We do not use other information from the message book. For example, we do not know whether a limit order is canceled or executed with another market order. Each x_t is a 1 by 20 vector and it contains the price and the number of shares of the lowest 5 ask limit orders and the price and the number of shares of the highest 5 bid limit orders. We have the data from 10:00 am to 16:00 pm. We use the data from 10:00 am to 1:00 pm for training and use the data from 1:00 pm to 4:00 pm for predicting and trading. We are going to ignore the training time as we mentioned early. Therefore we can use the model right after the training is done.

- Training
  - Data

    At each training step, we use the most recent 200 data points $x_t, x_{t+1}, x_{t+2}, \ldots, x_{t+199}$ to generate the input and target. x_{t+199} is the data at current time and x_t is the oldest. The input contains three matrices: price and volume.

$$price_{raw} = \begin{bmatrix} bp5_t, bp4_t, bp3_t, bp2_t, bp1_t, ap1_t, ap2_t, ap3_t, ap4_t, ap5_t \\ bp5_{t+1}, bp4_{t+1}, bp3_{t+1}, bp2_{t+1}, bp1_{t+1}, ap1_{t+1}, ap2_{t+1}, ap3_{t+1}, ap4_{t+1}, ap5_{t+1} \\ \vdots \\ bp5_{t+99}, bp4_{t+99}, bp3_{t+99}, bp2_{t+99}, bp1_{t+99}, ap1_{t+99}, ap2_{t+99}, ap3_{t+99}, ap4_{t+99}, ap5_{t+99} \end{bmatrix}$$

    where bpi_t is the ith best bid price at time t and api_t is the ith best ask price at time t. This is the raw data. We need to preprocess it before we feed it to the model. We calculate the mean of each column. Then for each element in $price_{raw}$, we subtract it by its column mean. After that the matrix we get is $price$.

$$volume_{raw} = \begin{bmatrix} bv5_t, bv4_t, bv3_t, bv2_t, bv1_t, av1_t, av2_t, av3_t, av4_t, av5_t \\ bv5_{t+1}, bv4_{t+1}, bv3_{t+1}, bv2_{t+1}, bv1_{t+1}, av1_{t+1}, av2_{t+1}, av3_{t+1}, av4_{t+1}, av5_{t+1} \\ \vdots \\ bv5_{t+99}, bv4_{t+99}, bv3_{t+99}, bv2_{t+99}, bv1_{t+99}, av1_{t+99}, av2_{t+99}, av3_{t+99}, av4_{t+99}, av5_{t+99} \end{bmatrix}$$

    where bpi_t is the ith best bid volume at time t and api_t is the ith best ask volume at time t. This is also the raw data. We need to preprocess it before we feed it to the model. First we apply natural log to every element in $volume_{raw}$. Then we calculate the mean of each column. For each element in $price_{raw}$, we subtract it by its column mean. After that the matrix we get is *volume*.

Now we need to introduce a new matrix called *capital*. It is the element wise product of *price* and *volume*. It is calculated inside the model. Matrix price and volume are the same for long model and short model. Target is different for long model and short model. Target is a scalar.

We want to explain the way we normalize the data. There are many ways to normalize data. Subtracting the mean is one of them. The most common way is Z-score [73]. What Z-score does is to subtract the element of each column by their mean and then divide them by their standard deviation. Then the gradient of each feature will be scaled to same level. That helps us avoid the zigzag gradient descent situation we have mentioned in the last chapter. The Z-score works great when we apply it to the entire dataset. It actually use the global mean and global standard deviation to normalize the data. However, in our experiment, we read data in time order. We do not know the global mean and standard deviation. We apply normalization on every 100 data points. Therefore, the effect of Z-score is limited. When we test it on the Apple data set from lobsterdata.com, the error, the AUC, and the accuracy of Z-score is still slightly better than subtracting the mean. The Adam optimization algorithm also helps close the gap because it considers momentum.

Table 4.2: Performance of Z-score and subtracting the mean

normalization	action	type	error	AUC	accuracy
subtracting the mean	long	train	0.5342	0.5457	0.7832
subtracting the mean	long	test	0.5520	0.4855	0.7723
subtracting the mean	short	train	0.5644	0.5402	0.7545
subtracting the mean	short	test	0.5721	0.5461	0.7415
Z-score	long	train	0.5305	0.5650	0.7791
Z-score	long	test	0.5451	0.5431	0.7720
Z-score	short	train	0.5637	0.5402	0.7536
Z-score	short	test	0.5860	0.4984	0.7396

The reason we choose to normalize the data by subtracting the mean is because normalizing data by Z-score gives us less profit(negative profit) in trading. Another benefit is that simply subtracting the mean is faster than Z-score normalization. Also the difference between two elements in any features does not change after subtracting the mean. We care about that more than the mean. Thus we still maintain the important information after the normalization.

  * Target for long model
    $Target = \begin{cases} 0 & \text{if } ap1_{t+99} + profit\ target_{train} > \max_{100 \le j \le 199} bp1_{t+j} \\ 1 & \text{otherwise} \end{cases}$
    we choose $profit\ target_{train}$ to be \$0.03.
  * Target for short model
    $Target = \begin{cases} 0 & \text{if } bp1_{t+99} < profit\ target_{train} + \min_{100 \le j \le 199} ap1_{t+j} \\ 1 & \text{otherwise} \end{cases}$
    we choose $profit\ target_{train}$ to be \$0.03. $profit\ target_{train}$ is defined early in this chapter.

– Model

The model is the same for long and short. The input is matrix price and volume. The size of matrix price, volume and capital are all 100 by 10. Even though the capital matrix is not a real input, it has the same size as the other two inputs. The information we want to capture is the change between each limit order record in time order. That is the difference of adjacent elements in a 2D matrix, which is exactly what convolutional neural network does. Therefore we choose to use convolutional neural network to capture the important feature from it. Therefore we can treat it as an "input" inside the model. We apply hidden layers with same structure to those three matrices before we concatenate them. The layers are:

  * 2D convolution layer. It has 16 filters and the size of each filter is 7 by 4. The activation function is relu function.
  * Max pooling layer. The pooling size is 3 by 3 and the stride is 3 by 3.
  * Faltten layer (It turns the input into a 1D vector).
  * Dense layer with dropout. The output of the dense layer is 1 by 8 vector. The dropout rate is 20%. The activation function is sigmoid function.

Even though we apply the layers with the same structure to matrix price, volume and capital, the parameters are not shared. After applying those layers to matrix price, volume and capital, we will get three 1 by 8 vectors respectively.

The next layer is to concatenate those three vectors to get a 1 by 24 vector.

The last hidden layer is a dense layer. The output is a scalar. The activation function is sigmoid function.

This scalar is the final output.

The following picture is our model.

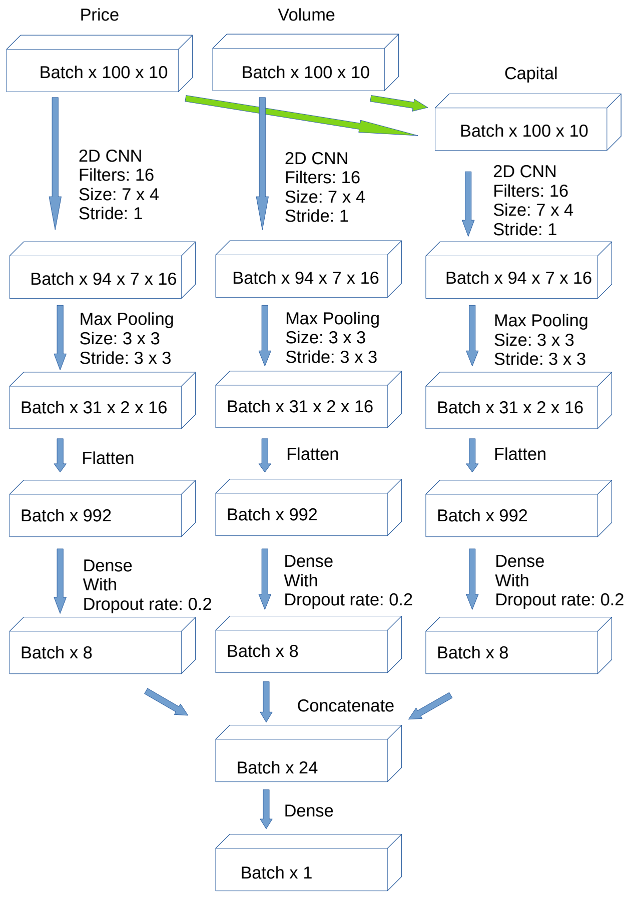


Figure 4.4: This is our trading model. The price matrix and the volume matrix are the input. The output is at the bottom which is a scalar. It has three convolution layers, three max pooling layers and three dropout layers.

  - Optimization function

    Here we use the Adam optimization algorithm. The learning rate is 0.001. $beta_1 = 0.9$ and $beta_2 = 0.999$.

  - Loss function

    The loss function is the cross entropy loss function.

  We use all the training data to train both model. The settings are also the same. We choose *batch size* to be 256 and *number of epoches* to be 1.

- Predicting

  - Data

    The data preprocessing is the same as the training except we use the data from 13:00 pm to 16:00 pm.

  - Output

    The last layer is similar to logistic regression. The output is a scalar between 0 and 1. We treat it as a probability as we describe in chapter 2. It indicates how likely there is a probability we can make profit $profit\ target_{train} = \$0.03$.

- Trading

  Before we trade, we need to set up a *probability threshold* $= 0.7$. If the probability we predict is higher than the *probability threshold*, we will open a position. If the probability from the model for long is higher than the *probability threshold*, we will long one share of the stock. If the probability from the model for short is higher than the *probability threshold*, we will short one share of the stock.

  The next step is to decide when to close the position. We will monitor the real best bid and ask price for 100 data points, which is 25 seconds. If there is a time that we can achieve the profit $profit\ target_{trade} = \$0.01$, we will close the position immediately. For example, we go long at time t, we pay $askPrice_t$ to buy a stock. If $bidPrice_{t+k} - askPrice_t \geq profit\ target_{trade}$, we sell the stock immediately. And this is very easy to implement, we just need to place a limit ask order at price $askPrice_t profit\ target_{trade}$. If we cannot achieve the profit goal in 25 seconds, then we sell it at the 25th seconds and we may lose money. The 25 seconds is the limit of holding time.

  Here is our trading strategy:

```
**if** position is opening **then**
  **if** profit $\geq profit\ threshold$ **or** *holding time* reach the limit **then**
```

```
    close position
  end if
else
  if predicted probability $\geq$ *probability threshold* then
    open position
  end if
end if
```

We can see that the profit is theoretically unlimited. It is possible that we lose money. We talked about setting up a profit bound early. Since a loose bound does not do much, we will try to set up a tight bound in the future.

4.7 Results and Analysis

The training is done on a server with a 6 Core i7-3770 CPU. We do not use GPU in this experiment. The code is written in Python. The major programming libraries we use are Tensorflow 2.0, numpy, matplotlib and scikit-learn. The code is hosted on Github: `https://github.com/FSUHeting/DL_LOB_Trading_and_MidPirce_Movement`. Each model takes 12 seconds to finish the training and it is negligible compared with our 3-hour trading period. We run the deep learning model 100 times and get the following results. The model involves a lot of randomnesses (random initialization and drop out), therefore, we want to use the average performance for evaluation.

- Profit for long only

Table 4.3: Profit for long only on Apple stock

Stock	Average	Maximum	Minimum	Positive rate	Standard deviation
Apple	\$0.02	\$0.13	\$0	70%	0.0191

- Profit for short only

Table 4.4: Profit for short only on Apple stock

Stock	Average	Maximum	Minimum	Positive rate	Standard deviation
Apple	\$0	\$0	\$0	0%	0

This is great! Our system never loses money. The 70% positive rate in table 4.3 means in 70 out of the 100 experiments the profit is 0. It means no trade happens during those experiments. 30 out of the 100 experiments the profit is positive. It means the system trades and makes a profit. The system for short actually does not trade at all. The reason is that none of the output probability is higher than the *probability threshold* (0.7). The following figure is the long opportunity we catch. There is a bid ask cross. We show the place we buy and sell one share of stock. As we have mentioned above, the model involves randomness. The places we buy and sell the stock are not fixed. That is why we only show a rough area where we buy and sell the stock.

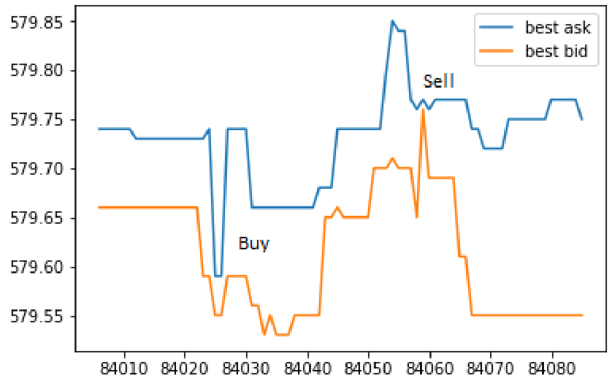


Figure 4.5: This is a success trading action on the Apple stock. We buy one share of Apple stock at a low price and then sell it at a high price. The time we buy and sell the stock is not fixed. There is some randomness in the model, so we run the experiment 100 times. Each time of our buying and selling the stock is slightly different.

Here are the results for other stocks.

- Profit for long only

Table 4.5: Profit for long only on Google, Amazon, Intel and Microsoft stock

Stock	Average	Maximum	Minimum	Positive profit rate	Standard deviation
Google	$0	$0	$0	0%	0
Amazon	$-0.3	$0	$-0.32	0%	0.0311
Intel	$0	$0	$0	0%	0
MSFT	$0	$0	$0	0%	0

- Profit for short only

Table 4.6: Profit for short only on Google, Amazon, Intel and Microsoft stock

Stock	Average	Maximum	Minimum	Positive profit rate	Standard deviation
Google	\$0.01	\$0.06	\$0	2%	0.0084
Amazon	\$0	\$0	\$0	0%	0
Intel	\$0	\$0	\$0	0%	0
MSFT	\$0	\$0	\$0	0%	0

We make less money from those stocks. For Intel and Microsoft(MSFT) stock, there is no trading at all. This is because of the imbalanced data problem [60] [61] we have mentioned early. Here is the positive sample rate for each stock at train and predict time. A sample here means 100 data points $x_t, x_{t+1}, x_{t+2}, \ldots, x_{t+99}$. A positive sample means we can make a profit in 100 *time interval*s. The positive sample rate means how much percentage of the time we can make a profit.

Table 4.7: Positive sample rate

Stock	train (long)	predict (long)	train(short)	predict(short)
Apple	0.217	0.228	0.240	0.259
Google	0.051	0.099	0.101	0.105
Amazon	0.108	0.111	0.127	0.111
Intel	0.006	0.003	0.016	0.005
MSFT	0.016	0.004	0.025	0.011

Since the positive sample rates of Intel and Microsoft stock are too low, the model will always predict do nothing. It has nothing to do with the *probability threshold*. Apple, Google and Amazon stocks are much better on positive sample rates. Therefore, the system will trade on those stocks. Since we set the *probability threshold* to be 0.7, the system does not trade much (less than 10 times). The system only does one short on Amazon stock. It happens right before the price jumps. The system cannot find a profit opportunity before the jump, so it waits until the end of 100 *time interval*s. Therefore, it loses money.

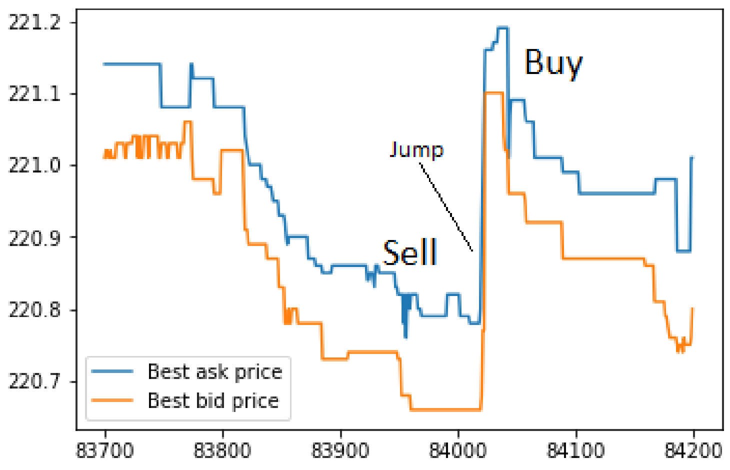


Figure 4.6: This is a fail trading accident on the Amazon stock. We sell one share of Amazon stock at a low price and then buy it back at a high price. The time we buy and sell the stock is not fixed. There is some randomness in the model, so we run the experiment 100 times. Each time of our buying and selling the stock is slightly different.

Table 4.8: Trading models performance on the testing dataset with different *probability threshold* and $profit\ target_{train}$.

action	*probability threshold*	$profit\ target_{train}$	average profit	average F1 score	average AUC	average precision	average recall
long	0.7	00.3	0.0192	0.00032	0.49151	0.10346	0.00016
short	0.7	00.3	0	0	0.53718	0	0
long	0.6	00.3	-0.0002	0.00128	0.48855	0.25801	0.00064
short	0.6	00.3	-0.0036	0.00006	0.53734	0.08545	0.00003
long	0.5	00.3	-0.1378	0.00387	0.48905	0.35372	0.00195
short	0.5	00.3	-0.2314	0.00074	0.53725	0.23668	0.00037
long	0.7	00.2	0.0286	0.00194	0.49608	0.72346	0.00097
short	0.7	00.2	0	0	0.53751	0	0
long	0.6	00.2	0.0128	0.00416	0.49663	0.73036	0.00208
short	0.6	00.2	-0.0202	0.00013	0.53673	0.10220	0.00006
long	0.5	00.2	-0.5658	0.01068	0.49864	0.62350	0.00541
short	0.5	00.2	-0.55	0.00246	0.53610	0.18581	0.00128
long	0.7	00.1	0.0288	0.00218	0.50535	0.85899	0.00109
short	0.7	00.1	0	0	0.53401	0	0
long	0.6	00.1	-0.0120	0.00635	0.50520	0.97543	0.00319
short	0.6	00.1	-0.0478	0.00043	0.53297	0.12884	0.00022
long	0.5	00.1	-0.4880	0.01168	0.240	0.84827	0.00590
short	0.5	00.1	-0.7582	0.00401	0.240	0.26260	0.00204

Figure 4.6 above demonstrates the price jump. It is rare, so we do not have much data about that. Therefore it is really hard for machine learning to detect that. Even though this is an accident, it does reveal a flaw in the system. That is we could have a big loss if there is a big

instant change in the price. There is a disadvantage of applying machine learning to real time financial time series. That is we cannot remove outliers before we predict. In order to make a profit in this situation, we want to be very confident before we make the trade. Therefore we want to set the $probability\ threshold = 0.7 > 0.5$ and $profit\ target_{train} = \$0.03 > \$0.01$. Thus we are more confident in the trading.

Finally, we can talk about the return rate. In financial investment, we need to consider the return rate. For the Apple stock, the initial funding is \$600. The maximum daily return is \$0.13 and the average daily return is \$0.02. Then the maximum daily return rate is 0.02166% and the average daily return rate is 0.00333%. Similarly, we get the return rate for Google and Amazon.

Table 4.9: Return rate for Apple, Google and Amazon

stock	initial funding	maximum return rate	average return rate
Apple	\$600	0.02166%	0.00333%
Google	\$600	0.01%	0.00166%
Amazon	\$250	-0.12%	-0.12%

Now we can compare it with the results of the most current paper [5]. The trading experiment is done on Lloyds Bank, Barclays, Tesco, BT, and Vodafone stocks. The data is from London Stock Exchange for the entire 2017 year. Since it is the data form the entire year and the authors did not mention the initial funding, we will use the approximate average stock price as the initial funding.

Table 4.10: Return rate for Lloyds Bank, Barclays, Tesco, BT and Vodafone

stock	average stock price	average return	average return rate
Lloyds Bank	£60	£0.00005	0.0000833%
Barclays	£200	£0.00015	0.0000750%
Tesco	£190	£0.00023	0.0001210%
BT	£330	£0.00030	0.0000909%
Vodafone	£200	£0.00013	0.0000650%

Despite we lose money on Amazon stock, the average return rates on Apple and Google stock are at least 10 times better than the results in table 4.10. We are improving the return rate to another level.

However, making a profit is not good enough. Investors always want the return rate to be at least higher than the risk free return rate. Namely, we want our return rate to be higher than the interest rate in the bank. Otherwise, it is not worth putting the money at risk. We can just take the interest from the bank. We will use the treasury bill rates as the risk free interest rate because the treasury bill is a U.S. government debt which is highly unlikely to default. The stock data is a daily stock data of June 21th 2012. The treasury bill rate on that day is

Table 4.11: Daily treasury bill rates data

	4 weeks	8 weeks	13 weeks	26 weeks	52 weeks
06/21/12	0.05	N/A	0.09	0.15	0.18

The data in table 4.11 above is retrieved from `https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=billRatesYear&year=2012`. We use the rate of 4 weeks(monthly) to calculate the daily rate.

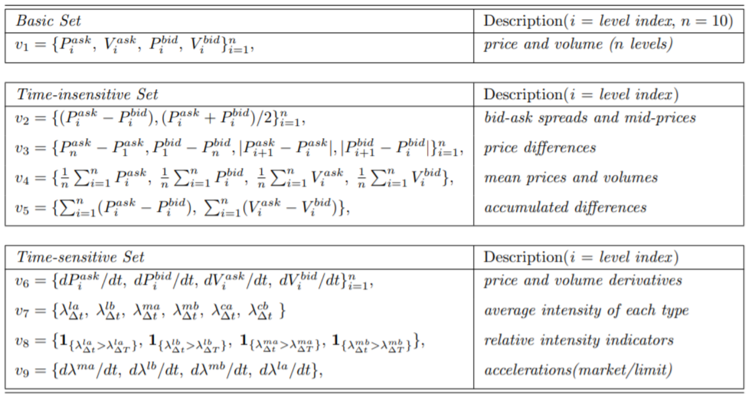
$$r = (1 + 0.05)^{\frac{1}{30}} - 1 = 0.1627\%$$

This is 8 times of our maximum daily return rate on Apple stock. That means even though we raise the return rate much by comparing with the previous paper [5], our model is still not ready for the industry. Our experiment is based on one day limit order book data. We only trade half a day because we only use the first half of the data for training. The experiment of the previous paper is done on one year limit order book data. Our experiment is just the beginning of the research on the limit order book data from Lobsterdata.com. We will gather more data for our future experiments. Then we will do more experiments and may improve the return rate even more.

CHAPTER 5

MID-PRICE MOVEMENT ON BENCHMARK DATASET

Mid-price movement is a highly popular experiment on limit order book data, especially on the following benchmark dataset. Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj and Alexandros Iosifidis (2018) [15] introduced a benchmark dataset. It can be downloaded at `https://etsin.avointiede.fi/dataset/urn-nbn-fi-csc-kata20170601153214969115`. The data source is the NASDAQ ITCH data file at Helsinki exchange from June 1, 2010 to June 14, 2010. It is actually 10 business days. The benchmark dataset wants to represent the stock market from different industries. Therefore, researchers pick the following five stocks: Kesko Oyj (Grocery Stores), Outokumpu Oyj (Steel), Sampo Oyj (Insurance), Rautaruukki Oyj (Steel) and Wartsila Oyj (Diversified Industrials). The ITCH file is a raw dataset not in the limit order book format. The researchers build the limit order book for those five stocks based on the ITCH file. After that, the researchers generate the following features [51] and add them into the benchmark dataset. However, we do not use the following features in our deep learning model.



Basic Set	Description(i = *level index*, $n = 10$)
$v_1 = \{P_i^{ask}, V_i^{ask}, P_i^{bid}, V_i^{bid}\}_{i=1}^n,$	*price and volume (n levels)*

Time-insensitive Set	Description(i = *level index*)
$v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^n,$	*bid-ask spreads and mid-prices*
$v_3 = \{P_n^{ask} - P_1^{ask}, P_1^{bid} - P_n^{bid}, \lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^n,$	*price differences*
$v_4 = \{\frac{1}{n}\sum_{i=1}^n P_i^{ask}, \frac{1}{n}\sum_{i=1}^n P_i^{bid}, \frac{1}{n}\sum_{i=1}^n V_i^{ask}, \frac{1}{n}\sum_{i=1}^n V_i^{bid}\},$	*mean prices and volumes*
$v_5 = \{\sum_{i=1}^n (P_i^{ask} - P_i^{bid}), \sum_{i=1}^n (V_i^{ask} - V_i^{bid})\},$	*accumulated differences*

Time-sensitive Set	Description(i = *level index*)
$v_6 = \{dP_i^{ask}/dt, dP_i^{bid}/dt, dV_i^{ask}/dt, dV_i^{bid}/dt\}_{i=1}^n,$	*price and volume derivatives*
$v_7 = \{\lambda_{\Delta t}^{la}, \lambda_{\Delta t}^{lb}, \lambda_{\Delta t}^{ma}, \lambda_{\Delta t}^{mb}, \lambda_{\Delta t}^{ca}, \lambda_{\Delta t}^{cb}\}$	*average intensity of each type*
$v_8 = \{\mathbf{1}_{\{\lambda_{\Delta t}^{la} > \lambda_{\Delta T}^{la}\}}, \mathbf{1}_{\{\lambda_{\Delta t}^{lb} > \lambda_{\Delta T}^{lb}\}}, \mathbf{1}_{\{\lambda_{\Delta t}^{ma} > \lambda_{\Delta T}^{ma}\}}, \mathbf{1}_{\{\lambda_{\Delta t}^{mb} > \lambda_{\Delta T}^{mb}\}}\},$	*relative intensity indicators*
$v_9 = \{d\lambda^{ma}/dt, d\lambda^{lb}/dt, d\lambda^{mb}/dt, d\lambda^{la}/dt\},$	*accelerations(market/limit)*

Figure 5.1: These are the handcrafted features from the limit order booking[51]. The features in the basic set are actually the order book. The features in the time insensitive set are generated by the order book. The features in the time sensitive set are generated by both the order book and the message book.

Finally, the researchers use three data normalization methods to preprocess the data and then share it online. The data normalization methods are:

- Z-score
  For each element in a column, subtract them by the mean of the column and then divide them by the standard deviation of the column.
- Min-Max
  For each element in a column, subtract them by the minimum of the column and then divide them by the difference of the maximum of the column and the minimum of the column.
- Decimal precision
  Divide each element in the dataset by 10^k, where k is the smallest integer that can make each normalized data less than 1.

The experiment is a 3-class classification task. We need to predict whether the average mid-price in 10 events is higher than or lower than or the same as the current mid-price. In order to accomplish the experiment, we need to modify the last layer of our model. Instead of one output we need the model to have three output: $y = (y_1, y_2, y_3)$. Also, we need to change the activation function from sigmoid function to softmax function:

$z = softmax(y)$ and $z_i = \frac{e^{y_i}}{\sum_{j=1}^{3} e^{y_j}}$ for $i = 1, 2, 3$.

Softmax function has two nice properties. First, $z_1 + z_2 + z_3 = 1$. Second, $z_i > z_j$ if $y_i > y_j$. Finally, we need to change the error function to multi class cross entropy.

$$Error = -(p_1 ln(z_1) + p_2 ln(z_2) + p_3 ln(z_3))$$

For p_1, p_2, p_3, only one of them is one and the other two are zero.

Since the benchmark dataset has data of 10 business days, the experiment is to test the model 9 times on this dataset. For the ith test, we use the first i days' data to train and use the $i + 1$th day's data to test. The final evaluation is based on the average performance of those 9 tests.

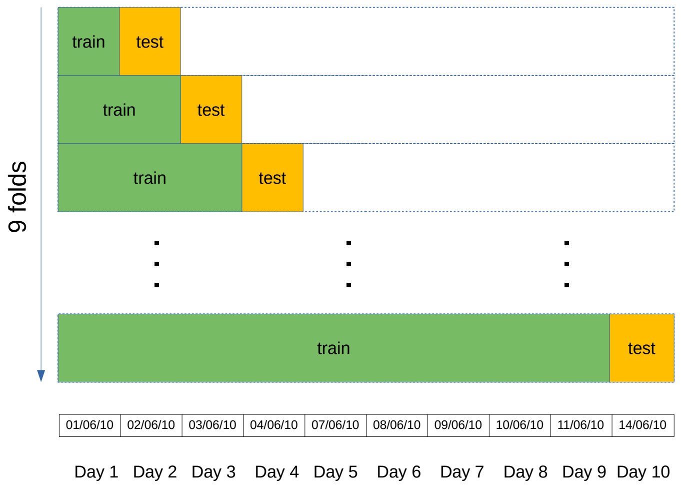


Figure 5.2: These are the 9 tests [15]. The benchmark dataset contains 10 days limit order book data. For test i, we use the first i days data for training and the $i + 1$th day's data for testing.

We pick the "Auction/3.Auction_DecPre" data in the dataset. It is preprocessed by using the decimal precision normalization. This is the easiest one of us to reconstruct the original limit order book data. From figure 5.1 we can see the features in the dataset. To reconstruct the order book, we only need the features on the basic set. Therefore we pick those features from the dataset. Then we need to multiply every element in the features by 10^k because they are preprocessed by using the decimal precision normalization. k will be determined by the format of the order book. According to LOBSTER_SampleFiles_ReadMe.txt from LobsterData.com, we know the stock price in the limit order book is multiplied by 10000. For example, if a ask price is \$ 53.21, then it will be 532100 in the limit order book. We can use this information to determine k. That is how we reconstruct the order book. Then we can apply our data preprocessing process and feed it to our deep learning model. However, the dataset does not contain information about time. Therefore we are doing an event based prediction instead of time based. Even though this experiment is different from our trading experiment and our model is not designed for this experiment, we still want to do a detailed comparison with all the experiments [15] [16] [17] that have been done based on the benchmark dataset.

Below is the results we get from our deep learning model. It is the average of the results of the 9 tests.

Table 5.1: The average performance of our regular mid-price model on the benchmark dataset

depth	accuracy	precision	recall	f1
3	0.311	0.247	0.335	0.176
5	0.317	0.346	0.345	0.211
10	0.339	0.353	0.347	0.243

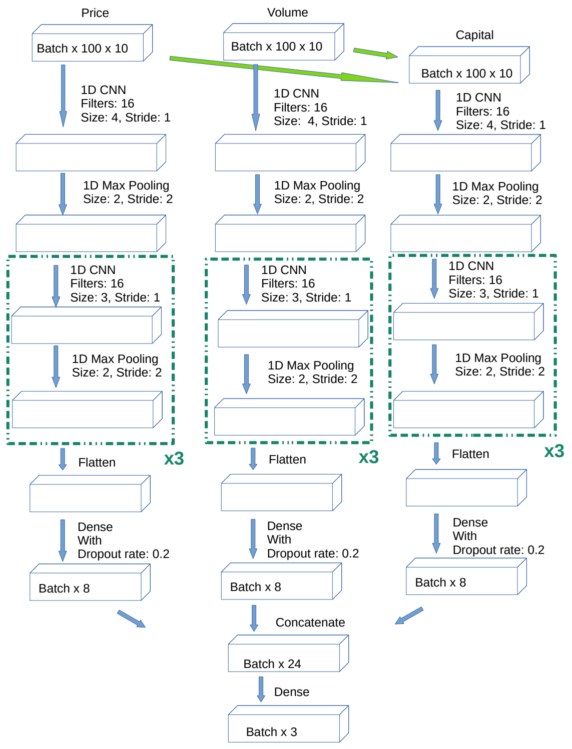


Figure 5.3: This is the benchmark mid-price model. It is very similar to the trading model. The differences are in the three green boxes. There are one CNN layer and a max pooling layer in the green box. These two layers in the green box will be repeated 3 times. That means there are 6 layers in total in each green box.

As you can see the performance is not very good. F1 score is very low. Therefore, we want to improve our model on mid-price experiment. Let's call the model above our regular mid-price model. Now we are going to introduce our benchmark mid-price model, which is a modification of our regular mid-price model. We are inspired by two papers [11] [7] about mid-price movement on the benchmark dataset. These two models introduce two deep convolution neural networks. We also want to make our model deeper. However, there are two problems in the papers and these two problems are quite common in the papers involving the benchmark dataset.

- Z-score data normalization
  The Z-score data normalization on the entire dataset needs future information, which is incorrect on time series. This problem is found in many papers [11] [7] [15] [17] [9] [14] using the benchmark dataset and it helps the models achieve great performance. Also the label in Z-score dataset is actually wrong.

- prediction highly biased on the majority classes
  The performances we find in many papers [11] [7] [15] [16] [17] [14] have a huge gap between the accuracy and the precision. This leads to a high accuracy but limits the f1 score.

We will talk about those two problems in details later. We pick the "Auction/3.Auction_DecPre" data in the dataset to avoid the Z-score problem. We want to make our benchmark mid-price model deeper than our regular mid-price model.

The table 5.2 is the performance of our benchmark mid-price model.

Table 5.2: The average performance of our benchmark mid-price model on the benchmark dataset

depth	accuracy	precision	recall	f1
3	0.467	0.477	0.449	0.448
5	0.464	0.478	0.449	0.445
10	0.467	0.481	0.449	0.447

However, this is not the full picture. The results below show how the benchmark mid-price model behaves at each test.

This is the test results from day 1 to day 9 by using a depth 3 limit order book data.

Table 5.3: Daily performance of our benchmark mid-price model on the depth 3 benchmark dataset

day	accuracy	precision	recall	f1
1	0.43862	0.42858	0.41344	0.41826
2	0.43659	0.48213	0.40910	0.39048
3	0.45299	0.45994	0.44678	0.44497
4	0.46720	0.48005	0.40735	0.39450
5	0.48514	0.50969	0.47154	0.48077
6	0.48438	0.48625	0.46178	0.46988
7	0.48711	0.48585	0.48378	0.48232
8	0.47381	0.48711	0.47388	0.47610
9	0.47296	0.47330	0.47442	0.47361

This is the test results from day 1 to day 9 by using a depth 5 limit order book data.

Table 5.4: Daily performance of our benchmark mid-price model on the depth 5 benchmark dataset

day	accuracy	precision	recall	f1
1	0.44416	0.43085	0.41137	0.40886
2	0.43201	0.48290	0.40550	0.37191
3	0.46170	0.47980	0.44045	0.45138
4	0.46679	0.45666	0.42778	0.41995
5	0.48091	0.49908	0.48064	0.47938
6	0.48847	0.49210	0.47161	0.47480
7	0.47182	0.48687	0.46787	0.46512
8	0.47155	0.48938	0.47151	0.47447
9	0.46093	0.48685	0.46174	0.45553

This is the test results from day 1 to day 9 by using a depth 10 limit order book data.

Table 5.5: Daily performance of our benchmark mid-price model on the depth 10 benchmark dataset

day	accuracy	precision	recall	f1
1	0.43120	0.43497	0.41749	0.40423
2	0.45813	0.46028	0.44743	0.44808
3	0.46506	0.47817	0.44573	0.45002
4	0.48143	0.47980	0.42330	0.43130
5	0.47661	0.51353	0.46093	0.46384
6	0.49338	0.50043	0.46230	0.47075
7	0.44914	0.48459	0.44430	0.42792
8	0.47717	0.48864	0.47699	0.46746
9	0.47399	0.48832	0.46484	0.46306

As you can see in table 5.3, 5.4, and 5.5, the overall performance roughly increases from test 1 to test 6 and decreases from test 6 to test 9. For example, in table 5.5, the accuracy increases 6 percent from test 1 to test 6, which is quite significant. However, the accuracy increases 2 percent from test 6 to test 9 in table 5.5, which is not significant. There are two reasons. One is that deep learning model may not have enough data at the beginning. Despite the papers [15] [16] [17] claim that the benchmark data contains about 4 million events, only 400 thousand events are found in the downloaded dataset, which is about the same number of events in the Apple stock sample data from www.Lobsterdata.com. Our benchmark mid-price model performs better when it is trained with more data. Another reason is that there is a distribution shift through days, which makes the performance of our benchmark mid-price model decreases a little bit at the 8th and 9th test. The following table is the three-class sample distribution from day 1 to day 9.

Table 5.6: Three-class sample distributions from day 1 to day 9

	Train			Test		
n	up	down	same	up	down	same
1	0.38465	0.37510	0.24023	0.41836	0.38452	0.19711
2	0.40078	0.37959	0.21961	0.38318	0.39681	0.21999
3	0.39636	0.38408	0.21955	0.39281	0.40992	0.19725
4	0.39551	0.39067	0.21381	0.42246	0.42310	0.15442
5	0.40072	0.39712	0.20215	0.38005	0.38478	0.23515
6	0.39705	0.39482	0.20812	0.41432	0.39265	0.19302
7	0.39975	0.39440	0.20584	0.33364	0.29116	0.37519
8	0.38883	0.37739	0.23376	0.33305	0.33434	0.33260
9	0.38117	0.37153	0.24729	0.37425	0.33540	0.29033

The training data components ratio is always about 2:2:1. From day 1 to day 9. The test data components ratio is about 2:2:1 from day 1 to day 6. From day 7 to day 9 the test data components ratio is about 1:1:1. It means that financial market changed since day 7 and we get different data from day 7 to day 9. Since we accumulate daily data for trainings, 3 days of new data do not change the components ratio in the training data much. Therefore, the training data and the testing data are from different distributions from day 7 to day 9. It means the training data and the testing data are from different financial market environments. Therefore, whatever our benchmark mid-price model learns from the training data may no longer be true during the testing. This explains why the benchmark mid-price model performance decreases for the last three days. This also makes the experiment on the benchmark dataset different from normal machine learning task, which the data distribution does not change much [81]. Therefore, we can easily optimize for the wrong model during the training.

Now we want to compare our results in table 5.2 with results in the three papers [15] [16] [17] of the benchmark dataset. The authors did a lot of experiments by using Ridge regression (RR) and SLFN network-based nonlinear regression on different data standardization methods.

- Ridge regression (RR)

  Ridge regression is a linear regression with L2 norm in error function. The model is exactly the same as the linear regression.

  $$y = wx + b$$

  where x is the input vector, y is the output scalar, w is the coefficient vector and b is the bias scalar. The error function is

  $$E = \sum_{i=1}^{n} (\hat{y} - y)^2 + \lambda ||w||_2$$

  where λ is a constant parameter. We need to set λ before calculating the error and the training. $\hat{y}$ is the target value of y in the data.

  For classification task in the paper [15], if y_i is the largest among the elements in y, the the model predicts it as class i.

- SLFN network-based nonlinear regression

  SLFN stands for a single layer feed forward network. It is a simple neural network with one hidden layer. The SLFN network-based nonlinear regression model is as follows:

  $$h = \phi_{RBF}(x)$$
  $$y = hw$$

  where the input x is a $n \times d$ matrix. n is the number of data points. d is the feature dimension. The output y is a $n \times c$ matrix. c is the number of class in the classification task in the paper [15]. w is a $k \times c$ coefficients matrix. h is a $n \times k$ matrix.

$\phi_{RBF}()$ is an element wised function

$$h_{ik} = e^{\dfrac{||x_i - v_k||_2}{2\sigma^2}}$$

where $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, k$. The v_k's are coefficients vectors. v is the matrix with k th column as v_k The error function is

$$E = ||y - \hat{y}||_2 + \lambda||w||_2$$

For classification task in the paper [15], if y_i is the largest among the elements in y, the the model predicts it as class i.

The tables below show the performance of model RR and SLFN on the benchmark dataset.

Table 5.7: Results of model RR and SLFN based on unfiltered representations

depth	RR Accuracy	RR Precision	RR Recall	RR F1
3	0.489	0.423	0.397	0.356
5	0.429	0.402	0.425	0.400
10	0.453	0.400	0.400	0.347
depth	SLFN Accuracy	SLFN Precision	SLFN Recall	SLFN F1
3	0.473	0.334	0.357	0.270
5	0.381	0.342	0.370	0.327
10	0.401	0.284	0.356	0.290

Table 5.8: Results of model RR and SLFN based on decimal precision normalization

depth	RR Accuracy	RR Precision	RR Recall	RR F1
3	0.490	0.432	0.386	0.330
5	0.435	0.406	0.430	0.405
10	0.451	0.417	0.399	0.349
depth	SLFN Accuracy	SLFN Precision	SLFN Recall	SLFN F1
3	0.504	0.465	0.421	0.393
5	0.457	0.451	0.449	0.438
10	0.461	0.453	0.420	0.399

Table 5.9: Results of model RR and SLFN based on min–max normalization

depth	RR Accuracy	RR Precision	RR Recall	RR F1
3	0.492	0.428	0.400	0.357
5	0.437	0.419	0.429	0.417
10	0.452	0.421	0.399	0.348
depth	SLFN Accuracy	SLFN Precision	SLFN Recall	SLFN F1
3	0.499	0.447	0.410	0.370
5	0.453	0.441	0.444	0.432
10	0.450	0.432	0.406	0.377

Table 5.10: Results of model RR and SLFN based on Z-score normalization

depth	RR Accuracy	RR Precision	RR Recall	RR F1
3	0.463	0.438	0.437	0.433
5	0.439	0.436	0.433	0.427
10	0.429	0.429	0.429	0.416
depth	SLFN Accuracy	SLFN Precision	SLFN Recall	SLFN F1
3	0.512	0.497	0.424	0.389
5	0.473	0.468	0.464	0.459
10	0.477	0.453	0.432	0.410

Our benchmark mid-price model is overall much better than the RR model and SLFM model on the unfiltered dataset and decimal precision normalized dataset. The red numbers in results of unfiltered representations and decimal precision normalization are the ones that better than our benchmark mid-price movement model. We will explain the reason later with the comparison with another paper [16]. Even though the results on min–max normalized dataset and Z-score normalized dataset are overall better than on unfiltered dataset and decimal precision normalized dataset, we cannot trust those results. The process of min–max normalization and Z-score normalization described in the paper [15] evolves using the global mean, global variance, global minimum and global maximum of the dataset. It means those normalization processes use future information, which is a serious problem in machine learning. That is the reason why results on min–max normalized dataset and Z-score normalized dataset appears to be better than on unfiltered dataset and decimal precision normalized dataset. The same mistake is made in another paper [17] and that is why we don't include the results here.

Another big problem in Z-score dataset is that the label is wrong. The test data is also normalized in the Z-score way. Then

$$Pirce_{ask(t)} + Price_{bid(t)} > Pirce_{ask(t+1)} + Price_{bid(t+1)}$$

does not equivalent to

$$\frac{Pirce_{ask(t)} - mean_{ask}}{std_{ask}} + \frac{Price_{bid(t)} - mean_{bid}}{std_{bid}} > \frac{Pirce_{ask(t+1)} - mean_{ask}}{std_{ask}} + \frac{Price_{bid(t+1)} - mean_{bid}}{std_{bid}}$$

It means that if we use the Z-score test dataset to calculate the label(mid-price movement), it can be wrong.

However, there is one[16] of the three papers paying attention to this issue. Instead of using global information to do normalization, the paper [16] used a rolling window to calculate mean and variance. The paper is focusing on improving f1 score by using the depth 10 dataset.

Table 5.11: Average performance on benchmark dataset from paper [16]

Sorting	Classifier	T	Accuracy	Precision	Recall	F1
Entropy	LMS	10	0.529	0.447	0.477	0.440
LMS1	LMS	10	0.540	0.437	0.456	0.430
LMS2	LMS	10	0.538	0.447	0.478	0.444
LDA1	LDA	10	0.616	0.408	0.398	0.397
LDA2	LDA	10	0.543	0.430	0.455	0.429
LDA1	LMS	10	0.604	0.468	0.431	0.408
LDA2	LMS	10	0.522	0.441	0.473	0.435
Entropy	RBFN	10	0.474	0.420	0.445	0.400
LMS1	RBFN	10	0.600	0.436	0.425	0.417
LMS2	RBFN	10	0.537	0.442	0.470	0.439
LDA1	RBFN	10	0.585	0.443	0.438	0.419
LDA2	RBFN	10	0.528	0.438	0.467	0.434

We can see that the f1 score is worse than the f1 score of our benchmark mid-price movement model in table 5.2. Therefore our benchmark mid-price movement model is a clear winner on the f1 score. However, the accuracy in the table above is much better than the accuracy of our benchmark mid-price movement model in table 5.2. It also happens in the first paper [15]. We want to address it since it is also the reason why the f1 score of our benchmark mid-price movement model is better than theirs. We can see that the results of the accuracy are much better than the results of precision (5% to 20% better). We need to explain how this is calculated first. The three papers [15] [16] [17] of the benchmark dataset only mention how to calculate the precision for binary classification problem. However, the experiment is actually a multi class classification problem. The two common ways to calculate precision for a multi-class classification problem are called "macro-averaged" and "micro-averaged" [62]. We choose "macro-averaged" to calculate the precision, recall and f1 score of our benchmark mid-price movement model. Apparently, the three papers [15] [16] [17] did not use "micro-averaged". We know accuracy and "micro-averaged" precision are the same thing. If those papers use the "micro-averaged" way, the accuracy and the precision will be the same.

Let's see how macro-averaged precision is calculated. Assume the following are the True positive and False positive for each class:

Table 5.12: True positive and False positive for each class

Class	TP	FP
up	U_{tp}	U_{fp}
down	D_{tp}	D_{fp}
same	S_{tp}	S_{fp}

Then the macro-averaged precision equals to

$$\frac{\frac{U_{tp}}{U_{tp}+U_{fp}}+\frac{D_{tp}}{D_{tp}+D_{fp}}+\frac{S_{tp}}{S_{tp}+S_{fp}}}{3}$$

The accuracy in this case equals to

$$\frac{U_{tp}+D_{tp}+S_{tp}}{U_{tp}+U_{fp}+D_{tp}+D_{fp}+S_{tp}+S_{fp}}$$

As you can see the macro-averaged precision only consider the precision in each class. If $\frac{U_{tp}}{U_{tp}+U_{fp}} \approx \frac{D_{tp}}{D_{tp}+D_{fp}} \approx \frac{S_{tp}}{S_{tp}+S_{fp}}$,

then
$\frac{U_{tp}}{U_{tp}+U_{fp}} \approx \frac{D_{tp}}{D_{tp}+D_{fp}} \approx \frac{S_{tp}}{S_{tp}+S_{fp}} \approx$ macro-averaged precision.
The absolute values of U_{tp}, D_{tp} and S_{tp} do not matter much.

However, in accuracy, the absolute values of U_{tp}, D_{tp} and S_{tp} matter. Assume

$U_{tp} = 500$, $U_{fp} = 500$, $D_{tp} = 500$, $D_{fp} = 500$, $S_{tp} = 50$, $S_{fp} = 50$.

Then both macro-averaged precision and accuracy are 50%. This is the case when the model is not biased on any class. Now assume the model is good at class U but bad at class S. We predict 50 more true positive in class U and 50 less true positive in class S. Now we change the outcomes to be

$U_{tp} = 550$, $U_{fp} = 450$, $D_{tp} = 500$, $D_{fp} = 500$, $S_{tp} = 0$, $S_{fp} = 100$.

Then accuracy is still 50% but the macro-averaged precision is 35%. This shows the accuracy is dominated by the true positives from the majority classes while the precision of each class is equally weighted in macro-averaged precision. This is why accuracy is not a good measure in machine learning with imbalanced data. If we consider the accuracy of the trivial model we have mentioned in chapter 6, the accuracy of that model is almost 100%. However, that trivial model does not know how to do a binary classification at all.

The reason that the results of the two papers have the accuracy much higher than the precision is that the machine learning algorithm is only good at detecting one or two out of the three classes. And those one or two classes are the majority classes. Since the paper [16] is focusing on optimize f1 score, this bias is a big problem. However, our benchmark mid-price movement model has the accuracy close to the precision. Thus, our benchmark mid-price movement model is not only the winner on the f1 score but also not biased on each class.

5.1 Mid-price Movement on Lobsterdata

The data sample from Lobsterdata is gaining popularity as an academic data sample [80] [79]. We want to evaluate both our regular mid-price model and our benchmark mid-price model on it. We can run the experiment and contribute some baseline performance for future research. Comparing to the benchmark dataset, the dataset from Lobsterdata contains information of time. This gives researchers freedom on preprocessing data and creating their own favorite features. We are going to set up the experiment as we do on the benchmark dataset except using the time base instead of the event base. We want to stick to our experimental design and use *time interval* = $0.25s$.

First, we want to predict the movement of the average future mid-price in 10 *time interval*s. There are three classes: up, down, same. The table 5.13 below is the distribution of the three classes of the training and testing data.

Table 5.13: Distribution of the three classes of the training and testing data

stocks	type	up	down	same
Apple	train	0.2294	0.2327	0.5378
Apple	test	0.1869	0.2015	0.6114
Google	train	0.1498	0.1720	0.6780
Google	test	0.1344	0.1562	0.7093
Amazon	train	0.2027	0.2271	0.5700
Amazon	test	0.1810	0.1933	0.6256

As you can see the distribution is dominated by the class "same". The stock price does not change much because 10 *time interval*s is short. From table 5.14 below, we can see there is the performance of our regular mid-price model. There is about a 20% gap between the accuracy and the precision.

Table 5.14: Performance of our regular mid-price model with 10 *time interval*s forecast range

stocks	type	accuracy	precision	recall	f1
Apple	train	0.6098	0.4052	0.3373	0.2664
Google	train	0.7089	0.4121	0.3341	0.2791
Amazon	train	0.6238	0.3909	0.3351	0.2638

Second, we want to predict the movement of the average future mid-price in 100 *time interval*s. There are three classes: up, down, same. Table 5.15 below is the distribution of the three classes of the training and the testing data.

As you can see in table 5.16 is the performance of our regular mid-price model. The gap between the accuracy and the precision is 15% now. The f1 score increases a little bit but the precision drops a lot.

Table 5.15: Distribution of the three classes of the training and the testing data

stocks	type	up	down	same
Apple	train	0.4345	0.4495	0.1158
Apple	test	0.3969	0.4691	0.1338
Google	train	0.3675	0.4599	0.1724
Google	test	0.3687	0.4549	0.1762
Amazon	train	0.4284	0.4745	0.0970
Amazon	test	0.4101	0.4388	0.1510

Table 5.16: Performance of our regular mid-price model with 100 *time interval*s forecast range

stocks	type	accuracy	precision	recall	f1
Apple	train	0.4430	0.2947	0.3353	0.2967
Google	train	0.4486	0.3601	0.3478	0.3007
Amazon	train	0.4185	0.2733	0.3251	0.2800

The experiment explains why the accuracy is not a good metric for machine learning with imbalanced data again. We can easily get a high accuracy for highly imbalanced data, which usually leads us to trivial models. We use the accuracy just to demonstrate the performance changes between different data distribution.

There is another reason why the performance of 10 *time interval*s and 100 *time interval*s are different. An important thing is to point out that each data point for the experiments is a (x, y) pair. However, the x does not change when we change the forecast range. Only the y changes, which means some x get reassigned to new classes. For example, we have 10 data points (x_1, y_1), (x_2, y_2), ..., (x_10, y_10).

In the first experiment, we are trying to find the difference between x_1 and x_2, x_3 so that we can classify them. There will be some features that can tell us the difference between them. In the second experiment, x_1, x_2 and x_3 are in the same group. Now we need to find the difference between x_1, x_2, x_3, x_4 and x_5, x_6, x_7, x_8. There is going to be some different features. Namely, we may need a different model or optimize the model in a different way. Our model is optimized for the trading task. The labels of the trading task and mid-price movement are quite different. This explains why our model gets low on f1 score.

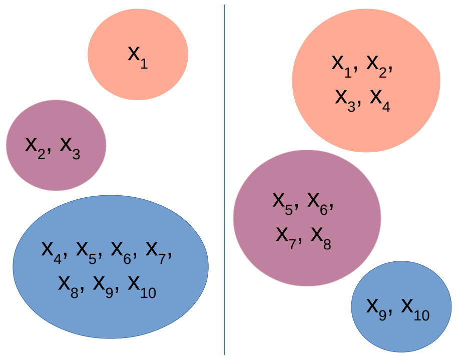


Figure 5.4: This is an example of the label change. When we choose different *time interval* or forecast range, the label of each data point changes. Since there are only three classes, it actually regroups the data.

Furthermore, if those data points are adjacent in time order, the difference between those points can be quite small. If the time interval is too small, some of the points may even be identical and it is impossible to find a feature to classify them. This could be a drawback of the time based model. Fortunately, we can control the time interval so that the drawback does not happen.

Since this data distribution of 100 *time interval*s is close to the benchmark dataset, we can also evaluate our benchmark mid-price model on it.

Table 5.17: Performance of our benchmark mid-price model with 100 *time interval*s forecast range

stocks	type	accuracy	precision	recall	f1
Apple	train	0.4458	0.2933	0.3378	0.3048
Google	train	0.4464	0.3151	0.3435	0.2926
Amazon	train	0.4249	0.2795	0.3300	0.2843

As you can see our benchmark mid-price model's performance is close to our regular mid-price model. This is not like what we have on the benchmark dataset. Our benchmark mid-price model is deeper than our regular mid-price model. That means there are more linear operations on the input matrices. That means we can potentially achieve high degrees of derivatives. The close performance of those two models indicates that the high degrees of derivatives may not be useful on the limit order book data from Lobsterdata.com. Since those two models' performances are close, we will pick the simpler one, our regular mid-price model, as the winner. A simpler model means less computation resources and easier to understand.

These results also tell us that those two limit order book dataset are quite different. Comparing to the performance of the same model on those two dataset may not be reasonable.

Finally, we want to talk about the overall poor performance. The f1 scores of all those experiments are below 0.5. The baseline here is actually $\frac{1}{3}$. Assume we have 100 data points. 40 of them are labeled up (U). 40 of them are labeled down (D). 20 of them are labeled same (S). Now, we will randomly guess the labels of those data points. We will guess 40% of their labels to be U, 40% of their labels to be D, and 20% of their labels to be S.

Then the precision and recall of the class U are both 0.4. The precision and recall of the class D are both 0.4. The precision and recall of the class S are both 0.2. Then the macro f1 score is

$$f1 = \frac{2\frac{0.4\times0.4}{0.4+0.4} + 2\frac{0.4\times0.4}{0.4+0.4} + 2\frac{0.2\times0.2}{0.2+0.2}}{3} = \frac{1}{3}$$

Therefore we are actually better than random guess. However, the f1 score is still not great. There could be two reasons. First, there could be noise data (uncommon data) in both training and testing dataset, especially in the testing dataset. The ultimate goal of mid-price movement prediction is to do real time prediction. Therefore, we will not know the label of the test data before we predict it. Thus, we do not know if the test data is noise or not. Second, the financial market changes over time so that there could be new data in the test dataset. The new data could be quite different from the training data. Therefore, the models cannot predict it correctly.

CHAPTER 6

CONCLUSION

We have successfully designed a deep learning model that can make a profit on Apple and Google stock. The first contribution of this thesis is that we improve the average daily return rate by more than 1000% comparing with the most current paper [5] on limit order book trading. This is a huge improvement on limit order book trading research. Even though the daily return rate is still lower than the risk free interest rate we get from US treasury bill, it demonstrates promising progress with only half day training data. We need to gather more data in the future. With more data, we can train a more sophisticated model, which could truly understand when to make trades.

The second contribution of this thesis is that we apply the mid-price movement experiment on the dataset from Lobsterdata.com. This is a new way of doing mid-price movement predictions since the dataset contains information of time. We also reveal a special problem in stock price prediction. The label of the data can be easily changed. For example, by choosing a different prediction range, the mid-price movement can switch from move up to remain the same. Therefore we may need different features or different models for each different experiment. The benchmark dataset [15], on the other hand, has predefined features but no information of time. This limits the creativity of the researchers since they cannot make any new features involve time. This thesis introduces a mid-price movement model as a new baseline model for future researchers.

We also apply our regular mid-price movement on the benchmark dataset [15]. The performance is not good. Therefore, we modify it to conduct the task better. Our benchmark mid-price model beats the models [15] [16] [17] on benchmark dataset. We also uncover one of the most meaningful issues in time series prediction, which is the distribution shift. The distribution shift makes the machine learning problem on the benchmark dataset different from the normal machine learning problem. The tests for the last three days may be meaningless. That is another reason we need to do the mid-price movement prediction on the dataset from Lobsterdata.com. Despite of this problem, we still beat the f1 score of the models on the benchmark dataset. We also point out another problem: using future data for data preprocessing [15] [17]. That is why good performances in those experiments should not be fully trusted. Data is very important in machine learning. Many papers on the limit order book we have discussed in this thesis ignoring the data analysis and tending to focus on the model design. The different performances of our models between two datasets also tells us that different dataset can lead to different model choice. This is the third contribution we have made in this thesis.

One of the work that will be done in the future is the risk control. For now, our trading model can only have a loose profit bound preventing us from losing huge amounts of money. We hope we can discover a way to provide a tighter profit bound with more data. That can also help us push the return rate to a higher level.

REFERENCES

[1] Zihao Zhang, Stefan Zohren, and Stephen Roberts (2018). BDLOB Bayesian Deep Convolutional Neural Networks for Limit Order Books. arXiv:1811.10041 [q-fin.CP].

[2] Arthur le Calvez and Dave Cliff (2018). Deep Learning can Replicate Adaptive Traders in a Limit-Order-Book Financial Market. arXiv:1811.02880 [cs.CE].

[3] Justin Sirignano (2016). Deep learning for limit order books. arXiv preprint arXiv:1601.01987.

[4] Justin Sirignano and Rama Cont (2018). Universal features of price formation in financial markets: perspectives from deep learning, arXiv preprint arXiv:1803.06917.

[5] Zihao Zhang, Stefan Zohren, and Stephen Roberts (2019). Extending Deep Learning Models for Limit Order Books to Quantile Regression. arXiv:1906.04404 [q-fin.TR]

[6] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2019). Deep Temporal Logistic Bag-of-features for Forecasting High Frequency Limit Order Book Time Series. ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)

[7] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2018). Using Deep Learning for price prediction by exploiting stationary limit order book features. arXiv preprint. arXiv:1810.09965.

[8] Jonathan Doering, Michael Fairbank, and Sheri Markose (2017). Convolutional Neural Networks Applied to High-Frequency Market Microstructure Forecasting. 2017 9th Computer Science and Electronic Engineering (CEEC).

[9] Zihao Zhang, Stefan Zohren, and Stephen Roberts (2018). DeepLOB Deep Convolutional Neural Networks for Limit Order Books. arXiv:1808.03668 [q-fin.CP].

[10] Daigo Tashiro, Hiroyasu Matsushima, Kiyoshi Izumi, and Hiroki Sakaji (2019). Encoding of high frequency order information and prediction of short term stock price by deep learning. Quantitative Finance Volume 19, 2019 - Issue 9: Machine Learning and AI.

[11] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2017). Forecasting Stock Prices from the Limit Order Book using Convolutional Neural Networks. 2017 IEEE 19th Conference on Business Informatics (CBI)

[12] Matthew F Dixon (2017). Sequence Classification of the Limit Order Book using Recurrent Neural Networks. arXiv:1707.05642 [q-fin.TR]

[13] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2019). Temporal Logistic Neural Bag-of-Features for Financial Time series Forecasting leveraging Limit Order Book Data. arXiv:1901.08280 [cs.LG]

[14] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2017). Using deep learning to detect price change indications in financial markets. 2511-2515. 10.23919/EUSIPCO.2017.8081663.

[15] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2018) Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. Journal of Forecasting 37.8 (2018), 852–866

[16] Adamantios Ntakaris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2019) Mid-price Prediction Based on Machine Learning Methods with Technical and Quantitative Indicators. arXiv:1907.09452

[17] Adamantios Ntakaris, Giorgio Mirone, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2019) Feature Engineering for Mid-Price Prediction with Deep Learning. arXiv:1904.05384

[18] Michael Negnevitsky (2015). Artificial intelligence: a guide to intelligent systems. Harlow, England ; New York : Addison-Wesley, 2005.

[19] Junwei Pan, Yizhi Mao, Alfonso L. Ruiz, Yu Sun, and Aaron Flores (2019). Predicting Different Type of Conversions with Multi-Task Learning in Online Advertising. SIGKDD (SIGKDD 2019)

[20] Ehtsham Elahi, Wei Wang, Dave Ray, Aish Fenton, and Tony Jebara (2019). Variational low rank multinomials for collaborative filtering with side-information. RecSys '19 Proceedings of the 13th ACM Conference on Recommender Systems Pages 340-347

[21] Rahul Goel, Shachi Paul, and Dilek Hakkani-Tur (2019). HyST: A Hybrid Approach for Flexible and Accurate Dialogue State Tracking. arXiv:1907.00883 [cs.CL].

[22] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis (2017). Mastering the game of Go without human knowledge. Nature volume550, pages354–359 (19 October 2017).

[23] Davi Frossard, Eric Kee, and Raquel Urtasun (2019). DeepSignals: Predicting Intent of Drivers Through Visual Attributes. To be presented at the IEEE International Conference on Robotics and Automation (ICRA), 2019.

[24] Menachem (Meni) Abudy and Avi Wohl (2018). Corporate Bond Trading on a Limit Order Book Exchange. Review of Finance, 2018, 1413–1440 doi: 10.1093/rof/rfx054

[25] Ioane Muni Toke and Nakahiro Yoshida (2016). Modelling intensities of order flows in a limit order book. Quantitative Finance, 2017 Vol. 17, No. 5, 683–701, http://dx.doi.org/10.1080/14697688.2016.1236210

[26] Xiaofei Lu and Frederic Abergel (2018). High-dimensional Hawkes processes for limit order books: modelling, empirical analysis and numerical calibration. Quantitative Finance, 2018 Vol. 18, No. 2, 249–264, https://doi.org/10.1080/14697688.2017.1403142

[27] Julius Bonart and Martin D. Gould (2017). Latency and liquidity provision in a limit order book. Quantitative Finance, 2017. Vol. 17, No. 10, 1601–1616, https://doi.org/10.1080/14697688.2017.1296177

[28] Dave Cliff (2018). An Open-Source Limit-Order-Book Exchange for Teaching and Research. 2018 IEEE Symposium Series on Computational Intelligence (SSCI)

[29] Ulrich Horst and Dorte Kreher (2018). Second order approximations for limit order books. Finance Stoch (2018) 22:827–877. https://doi.org/10.1007/s00780-018-0373-7

[30] Lei Li, Yihang Ou, Yabin Wu, Qi Li, and Daoxin Chen (2018). Research on feature engineering for time series data mining. 2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC)

[31] Sergios Theodoridis and Konstantinos Koutroumbas (2009). Pattern recognition 4th ed. Academic Press; 4 edition (November 3, 2008). ISBN-10: 1597492728, ISBN-13: 978-1597492720.

[32] Darrell B. Vincent (2018). Pattern Recognition: Practices, Perspectives and Challenges. Nova Science Pub Inc; UK ed. edition (June 30, 2013). ISBN-10: 1626181969, ISBN-13: 978-1626181960

[33] Sankar K. Pal and Amita Pal (2001). Pattern Recognition: From Classical to Modern Approaches. World Scientific, 2001. ISBN 9810246846, 9789810246846.

[34] Michal Krol and Magdalena Krol (2019). Learning From Peers' Eye Movements in the Absence of Expert Guidance: A Proof of Concept Using Laboratory Stock Trading, Eye Tracking, and Machine Learning. Cognitive ScienceVolume 43, Issue 2

[35] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid (2018). Practical Deep Reinforcement Learning Approach for Stock Trading. arXiv:1811.07522

[36] Felipe Dias Paiva, Rodrigo Tomas Nogueira Cardoso, Gustavo Peixoto Hanaoka, and Wendel Moreira Duarte (2019). Decision-making for financial trading: A fusion approach of machine learning and portfolio selection. Expert Systems with Applications Volume: 115 Issue 1 (2019) ISSN: 0957-4174

[37] Dongdong Lv, Shuhan Yuan, Meizi Li, and Yang Xiang (2019). An Empirical Study of Machine Learning Algorithms for Stock Daily Trading Strategy. Mathematical Problems in Engineering Volume 2019, Article ID 7816154, 30 pages. https://doi.org/10.1155/2019/7816154

[38] Rajashree Dasha and Pradipta Kishore Dashb (2016). A hybrid stock trading framework integrating technical analysis with machine learning techniques. The Journal of Finance and Data Science Volume 2, Issue 1, March 2016, Pages 42-57.

[39] Sang-Soo Park, Jung-Hyun Hong, and Ki-Seok Chung (2017). Modified Convolution Neural Network for Highly Effective Parallel Processing. 2017 IEEE International Conference on Information Reuse and Integration (IRI) IRI Information Reuse and Integration (IRI), 2017 IEEE International Conference on. :325-331 Aug, 2017

[40] Tao Gong, Tiantian Fan, Jizheng Guo, and Zixing Cai (2015). Gpu-based parallel optimization and embedded system application of immune convolutional neural network. 2015 International Workshop on Artificial Immune Systems (AIS) Artificial Immune Systems (AIS), 2015 International Workshop on. :1-8 Jul, 2015

[41] Che-Lun Hung, Yi-Yang Lin, Chuan Yi Tang, Chilung Wang, and Ming-Chiang Chen (2018). Performance of Convolution Neural Network based on Multiple GPUs with Different Data Communication Models. 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2018 19th IEEE/ACIS International Conference on. :87-92 Jun, 2018

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2017). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM Volume 60 Issue 6, June 2017 Pages 84-90 ACM New York, NY, USA. DOI: 10.1145/3065386

[43] Keiron O'Shea and Ryan Nash (2015). An Introduction to Convolutional Neural Networks. arXiv:1511.08458v2

[44] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi (2017). Understanding of a convolutional neural network. 2017 International Conference on Engineering and Technology (ICET). DOI: 10.1109/ICEngTechnol.2017.8308186

[45] Omer Berat Sezer and Ahmet Murat Ozbayoglu (2018). Algorithmic financial trading with deep convolutional neuralnetworks: Time series to image conversion approach. TOBB University of Economics and Technology, Ankara 06560, Turkey.

[46] Lina Nia, Yujie Lia, Xiao Wanga, Jinquan Zhanga, Jiguo Yud, and Chengming Qi (2018). Forecasting of Forex Time Series Data Based on Deep Learning. 2018 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2018.

[47] Chien-Liang Liu, Wen-Hoar Hsaio, and Yao-Chung Tu (2019). Time Series Classification With Multivariate Convolutional Neural Network. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 66, NO. 6, JUNE 2019

[48] Anastasia Borovykh, Sander Bohte, and Cornelis W. Oosterlee (2018). Conditional time series forecasting with convolutional neural networks. arXiv:1703.04691v5

[49] Anthony Brunel, Johanna Pasquet, Jerome Pasquet, Nancy Rodriguez, Frederic Comby, Dominique Fouchez, and Marc Chaumont (2019). A CNN adapted to time series for the classification of Supernovae. Color Imaging XXIV: Displaying, Processing, Hardcopy, and Appli., IS&T Int. Symp. on Electronic Imaging, SF, California, USA, 13 - 19 Jan. 2019.

[50] Ding Gang, Lei Da and Zhong Shisheng (2014). Time series prediction using convolution sum discrete process neural network. DOI: 10.14311/NNW.2014.24.025.

[51] Alec N. Kercheval and Yuan Zhang (2015). Modelling high-frequency limit order book dynamics with support vector machines. Quantitative Finance, vol 15, no 8, 2015, 1315–1329.

[52] Myron Scholes and Joseph Williams (1977). "Estimating betas from nonsynchronous data". Journal of Financial Economics. 5: 309–327. doi:10.1016/0304-405X(77)90041-1.

[53] Mark C. Lundin, Michel M. Dacorogna, and Ulrich A. Muller (1999). Chapter 5: Correlation of High Frequency Financial Time Series". In Pierre Lequex (ed.). The Financial Markets Tick by Tick. pp. 91–126.

[54] Takaki Hayashi and Nakahiro Yoshida (2005). "On covariance estimation of non-synchronously observed diffusion processes". Bernoulli. 11: 359–379. doi:10.3150/bj/1116340299.

[55] Kira Rehfeld, Norbert Marwan, Jobst Heitzig, and Jurgen Kurths (2011). Comparison of correlation analysis techniques for irregularly sampled time series. Nonlinear Processes in Geophysics. 18: 389–404. doi:10.5194/npg-18-389-2011.

[56] Zihao Zhang, Stefan Zohren and Stephen Roberts (2019). Extending Deep Learning Models for Limit Order Books to Quantile Regression arXiv preprint arXiv:1906.04404v1.

[57] Christopher M. Bishop (2011). Pattern Recognition and Machine Learning. Springer (April 6, 2011). ISBN-13: 978-0387310732.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). Deep Learning. MIT Press 2016.

[59] Balazs Csanad Csaji (2001) Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary

[60] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard (2004). A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations, 6(1).

[61] Eric Bauer and Ron Kohavi (1999). An empirical comparison of voting classification algorithms: Bagging, boosting and variants. Machine Learning, 36(1,2).

[62] Vincent Van Asch (2013). Macro-and micro-averaged evaluation measures [[BASIC DRAFT]].

[63] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. Bibcode:1986Natur.323..533R. doi:10.1038/323533a0.

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS 2012)

[65] Karen Simonyan and Andrew Zisserman (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[66] Mohamed A. El-Sayed, Yarub A. Estaitia, and Mohamed A. Khafagy (2013). Automated Edge Detection Using Convolutional Neural Network. International Journal of Advanced Computer Science and Applications(IJACSA), Volume 4 Issue 10, 2013.

[67] Aneeqa Ahmed and Yung-Cheol Byun (2019). Edge Detection using CNN for Roof Images. APIT 2019 Proceedings of the 2019 Asia Pacific Information Technology Conference Pages 75-78.

[68] Prewitt, J.M.S. (1970). Object Enhancement and Extraction. Picture Processing and Psychopictorics, B. Lipkin and A. Rosenfeld, Eds., New York: Academic Press, 1970, pp. 75-149.

[69] Hany Farid and Eero P. Simoncelli (2004), Differentiation of discrete multi-dimensional signals, IEEE Trans Image Processing, vol.13(4), pp. 496–508, Apr 2004.

[70] F N Hussin, H A Rahman, and A Bahar (2017). Stochastic differential equation (SDE) model of opening gold share price of bursa saham malaysia. Journal of Physics: Conference Series, Volume 890, conference 1.

[71] Thomas Chinwe Urama and Patrick Oseloka Ezepue (2018). Stochastic Ito-Calculus and Numerical Approximations for Asset Price Forecasting in the Nigerian Stock Market. Journal of Mathematical Finance, 2018, 8, 640-667.

[72] Daniel P. Collins (2017). The quants take on fintech. Modern Trader. May2017, p24-27. 4p.

[73] W. Bilal Khan (2018). Differences Between Standard Deviation and Robust NIQR Method During Z-Score Evaluation. 2018 15th International Conference on ElectroMagnetic Interference and Compatibility (INCEMIC) 13-16 Nov. 2018.

[74] Alan Agresti (2002). Categorical Data Analysis. New York: Wiley-Interscience. ISBN 978-0-471-36093-3.

[75] Takeshi Amemiya (1985). "Qualitative Response Models". Advanced Econometrics. Oxford: Basil Blackwell. pp. 267–359. ISBN 978-0-631-13345-2.

[76] David R. Cox (1966). "Some procedures connected with the logistic qualitative response curve". In F. N. David (1966) (ed.). Research Papers in Probability and Statistics (Festschrift for J. Neyman). London: Wiley. pp. 55–71.

[77] Kevin Murphy (2012). Machine Learning: A Probabilistic Perspective. MIT. ISBN 978-0262018029.

[78] Bolin Gao and Lacra Pavel (2017). On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning. arXiv:1704.00805

[79] Markus Bibinger, Christopher Neely, and Lars Winkelmann (2019). Estimation of the discontinuous leverage effect: Evidence from the NASDAQ order book. Journal of Econometrics Volume 209, Issue 2, April 2019, Pages 158-184. https://doi.org/10.1016/j.jeconom.2019.01.001

[80] Markus Bibinger, Moritz Jirak, and Markus Reiß (2015). Volatility estimation under one-sided errors with applications to limit order books. arXiv:1408.3768 [math.PR]

[81] Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2 edition. Springer Series in Statistics book series (SSS) (August 26, 2009).

[82] Heman Mohabeer, K.M. Sunjiv Soyjaudah, and Narainsamy Pavaday (2011). Enhancing the performance of neural network classifier using selected biometricfeatures. SENSORCOMM 2011 : The Fifth International Conference on Sensor Technologies and Applications.

[83] Tom Fawcett (2006). An Introduction to ROC Analysis. Pattern Recognition Letters. 27 (8): 861–874. doi:10.1016/j.patrec.2005.10.010.

[84] David M W Powers (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. Journal of Machine Learning Technologies. 2 (1): 37–63.

BIOGRAPHICAL SKETCH

Heting Yan obtained his B.S. degree in applied mathematics from Lanzhou University in Lanzhou, China in 2011, M.S. degree in applied mathematics from University of Colorado Denver in 2013, and PhD degree in financial mathematics from Florida State University in 2020. His research focuses on deep learning and limit order book. He is also interested in big data, natural language processing, and recommendation system.