Leverage Financial News to Predict Stock Price Movements Using Word Embeddings and Deep Neural Networks

Yangtu Peng

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

Graduate Program in Lassonde School of Engineering Department of Electrical Engineering and Computer Science York University Toronto, Ontario April 2016

© Yangtu Peng, 2016
Abstract

With the booming of deep learning in the recent decade, deep neural network has achieved state-of-art performances on many machine learning tasks and has been applied to more and more research fields. Stock market prediction is an attractive research topic since the successful prediction on the market’s future movement leads to significant profit. In this thesis, we investigate to combine the conventional stock analysis techniques with the popular deep learning together and study the impact of deep neural network on stock market prediction.

Traditional short term stock market predictions are usually based on the analysis of historical market data, such as stock prices, moving averages or daily returns. Whereas financial news also contains useful information on public companies and the market. In this thesis we apply the popular word embedding methods and deep neural networks to leverage financial news to predict stock price movements in the market. Experimental results have shown that our proposed methods are simple but very effective, which can significantly improve the stock prediction accuracy on a standard financial database over the baseline system using only the historical price information.
Acknowledgements

I would like to express my gratitude to all those who gave me the chance to complete this thesis. I am deeply indebted to my supervisor Prof. Hui Jiang whose help, stimulating suggestions and encouragement helped me in all the time of the research and writing of this thesis. Many thanks are also given to the thesis committee member Prof. Peter Lian, Prof. Jack ZhenMing Jiang, and Prof. Yun Gao. In addition, my heartfelt thanks to my many friends at York University.
Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Tables vi

List of Figures vii

Abbreviations ix

1 Introduction 1
   1.1 Motivations of this work 2
   1.2 Related works 5
   1.3 Goals of this thesis 8
   1.4 Organizations of this thesis 9

2 Deep Neural Networks 11
   2.1 Structure of feedforward neural network 12
   2.1.1 Network's Input 14
   2.1.2 Feed-forward network function 14
   2.1.3 Activation function 17
   2.1.4 Network's output 19
   2.2 Training DNN 20
   2.2.1 Loss function 20
   2.2.2 Stochastic Gradient Descent 22
   2.2.3 Error Backpropagation 25

3 Word Embedding 30
   3.1 Language modeling 31
   3.2 Learning distributed representation of words 35

4 Financial Modeling 40
   4.1 Traditional stock market prediction 42
   4.2 Prediction with neural networks 45
List of Tables

5.1  List of fields in each daily security record queried from CRSP database. . 51
6.1  An example of financial news data entry. ................................. 61
6.2  List of news title published on 2007-01-09. ............................... 68
8.1  Experiments on different input timeframes. We show that given other parameter unchanged, the input samples containing 5 days’ data generates the best performance. .............................................. 79
8.2  Experiments on different DNN structures. Number of elements in the square brackets represent the number of hidden layers used in the neural networks, the value of each element denotes the number of hidden units in the corresponding hidden layer. Even though the networks with 3, 4 or 5 layers of 1024 hidden units generates almost the same performance, we choose the structure of 3 layers as the optimal one as it has the least computational cost. .......................................................... 80
8.3  Experiments on different learning rates. We show that with validation, network with learning rate set to 0.005 produces the best performance and converge fast. ......................................................... 80
8.4  Experiments on different mini-batch size. .................................. 80
8.5  Stock prediction error rates on the test set. ................................. 81
8.6  2 x 2 contingency table applied in McNemar’s test. ....................... 82
8.7  2 x 2 contingency table of the results from random classifier and the DNN model with price data. ................................................. 83
8.8  Results of the McNemar’s Test of our models. ............................ 84
List of Figures

1.1 Illustration of the volatility of stock market. .................. 3

2.1 An simple example of neural network with a single hidden layer. Each circle represents a computing unit in the network. Each arrow represents the connection between two units. The direction of arrow indicates that the input information is propagated from input layer at the bottom to the output layer at the top. .................. 12

2.2 Two examples of activation functions. .................. 18

2.3 Overall procedure of error back propagation algorithm. Blue arrow indicates the direction of feed forward networks flow from input layer to output layer. Red arrows indicates the direction of backward error propagation. .................. 29

3.1 Architecture of two models for learning high quality word embedding proposed in [1]. Both models use the word sequence \{x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}\} as an example. CBOW takes \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\} as input and predict the middle word \(x_t\). Whereas skip-gram takes the middle word \(w_t\) as input and predict the surrounding context \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\}. .. 36

5.1 Example of exponential moving average of Google’s close price during 2006-01-03 and 2006-05-25. The fluctuation in original data is smooth out in EMA. .......................... 56

5.2 Example of relative strength index of Google’s close price. The region between 30 and 70 is typically the neutral region. RSI higher then 70 indicates an overbought of the stock and price drop is about to happen, whereas RSI lower then 30 indicates an oversold of the stock and price raise is about to happen. .......................... 57

6.1 Parsing result of the example “Apple slipped behind Samsung and Microsoft in a 2013 customer experience survey from Forrester Research”. .......................... 67

7.1 An example shows a pair of stocks which are highly related to each other. The two graphs above plot the closing price of ”CISCO SYSTEM INC” and ”BUCKLE INC” during the period of 2006-01-03 to 2013-12-31. .. 72

7.2 Illustration of a part of correlation graph which contains 11 stocks. The symbol in the circle are the ticker name of the stock. The value along the edges are the correlation score of the two stocks connected by that edge. 73

8.1 Probability density function of Chi-squared distribution. \(k\) denotes the degree of freedom. .......................... 82
8.2 Predict unseen stocks via correlation ................. 85
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>NLP</td>
<td>Nature Language Processing</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>BoW</td>
<td>Bag of Words</td>
</tr>
<tr>
<td>EMA</td>
<td>Exponential Moving Average</td>
</tr>
<tr>
<td>RSI</td>
<td>Relative Strength Index</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Term Frequency-Inverse Document Frequency</td>
</tr>
<tr>
<td>PMI</td>
<td>Pointwise Mutual Information</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The concept of artificial neural network was introduced decades ago as a machine learning model. However, due to the limited training data available, as well as the limitation of computational power back then, it was expensive and inefficient to implement this technique into a real life application. Whereas other simpler machine learning models such as support vector machine (SVM) and linear classifier was more popular in the research of machine learning problems, and most of the state-of-art performance were achieved by those simpler models.

Over the past few years, with the emergence of big data era, both researchers and the businesses are looking to make use of the explosively growing data. Furthermore, the computing power of modern systems has been significantly improved by the advance in hardware development, especially with the aid of powerful graphic processing units (GPU), the training time of neural network have been largely reduced thanks to GPU’s fast computation among matrices or vectors. These changes created more possibilities and opportunities of applying neural network in real life applications. In later 2000s, professor Geoffrey Hinton’s publication of how to efficiently train a multilayer feed-forward
neural network [12] has brought back the interest of neural networks to the community of machine learning. After that, more and more studies have been published to show that with a big amount of data and bigger neural network models, combining with the computing power of modern GPUs, it is possible for neural networks to outperform the simpler models and achieve state-of-art performance in many traditional machine learning tasks. For example, in [4–7], it has been shown that by simply replacing the linear model of parser in syntactic parsing problems with a fully connected feed-forward network, a better performance could be achieved. These lead to the resurgence of neural network with the term “deep learning”, where “deep” refers to the bigger number of hidden layers in neural network’s architecture.

1.1 Motivations of this work

Financial data modelling is the problem of constructing an abstract representation of the real world financial information. One of the most important purpose of this task is to give people a predictive vision on the performance or valuations of certain asset base on the available information, and help them to make financial decisions which eventually generate profits. It has a wide range of applications in the business world, for examples, accounting, corporate finance, investment banking and stock valuations. Stock market is an attractive place for investor as it generates relatively higher returns comparing to other conservative investments. However the volatility of stock market also makes it a risky place to survive, for instance, figure 1.1 shows how price of a stock could drop or raise nearly 10% in a single trading day. Therefore, being able to successively predict the future movement of a company’s stock price can not only yield significant profit, but also eliminate the potential lost for the investors.
Even though the well-known efficient markets hypothesis [16] suggest the movement in stock market is not predictable, financial analysts have never stopped exploring new method to counter this viewpoint, and it has been a long-lasting argument whether it is possible to predict future stock prices only based on the past price information. With the advent of information age, stock analysts have moved their interest into modern computers and are looking forward to creating more accurate predictive methods that rely on the advanced computing technologies.

Nowadays, one of the most prominent techniques is artificial neural network. Recently, deep neural networks (DNNs) have achieved huge successes in many data modelling and prediction tasks ranging from speech recognition, computer vision to natural language processing. For examples, [2, 37, 39, 40] show significant improvements on the word-error rate of speech-to-text transcription task, which was contributed by the effective and
efficient modelling ability of a deeper neural network architecture as well as a large set of training data; [3] proposed a large deep convolutional neural network for high-resolution images’ classification problem and achieved a result that is relatively better than previous state-of-art performance; moreover, [32, 38] presented a new recurrent neural network based language model that the results have shown significant improvements on both language modelling task as well as speech recognition problems comparing to the state-of-art language model.

In addition to these experimental facts, artificial neural network has also been proven to be powerful in theory. It was first showed in [10], that by using a discriminatory activation function, such as the continuous sigmoidal functions, as nonlinearity, a multi-layer feed-forward neural network with a single hidden layer containing finite number of neurons could uniformly approximate any arbitrary continuous functions. Later in [11], it has been established that it was the multilayer feed forward architecture of neural network that gives it the ability of universally approximating arbitrary continuous functions. That means as long as the neural network contains sufficient amount of hidden units (the neurons), even with mild assumption on the activation function, it can approximate arbitrary continuous functions. Furthermore, [8] states that a simple neural network could be use to learn a wide variety of problems if it is trained with appropriate inputs.

Previous works have shown the impressive computational power of neural network in not only the traditional machine learning tasks, but also the possibility of applying neural networks as universal approximators into other new problems. In this thesis, we are interested in applying this powerful deep learning method to the problem of financial data modelling, in particular, to predict stock price movements.
1.2 Related works

Financial institutions have started to support researches in financial forecasting using neural networks decades ago. Many studies demonstrates the applicability of predicting financial and economic time series either empirically or theoretically. For instances, [14] conducted a review of a set of studies on how efficient are neural networks in varies type of real-life tasks. The reviewed studies includes different types of business forecast and prediction, for instance, stock price prediction, financial distress pattern recognition, or bond rating. This review employs a set of criteria, called *effectiveness of validation* and *effectiveness of implementation*, to evaluate the selected studies in order to verify whether the proposed method is applicable to the targeted problem. It concludes that neural networks with efficient implementation and well-defined validation process show potential for business forecast, and neural networks have already outperformed other simpler but well accepted models in some tasks. Moreover, there is research summa-

rizing general instructions on how to design such a neural network system for financial forecasting; [13] discussed the general parameters of a neural network prediction system and the difficulty of developing and tuning such system; base on the nature of economic forecasting problem as well as the convention of neural network development, it proposed a design procedure to provide an efficient way of constructing a neural network system for financial market prediction.

Traditionally neural networks have been used to model stock data as time series for the forecasting purpose. In a real financial market, there are many factors that could be used as criteria to evaluate the performance of a forecasting system, such as stock price of an individual company, return of a portfolio, or the price value of a market index. Therefore different researches tried to establish their proposed model based on
different evaluation standards. For example, [15] established a neural network that is trained as a linear regression model to predict the next day’s closing price of a specific stock. It used ten technical indicators to construct its input feature vectors. In this work, the mean square error was used as the error function to train the network and evaluate the final performance of the model. Whereas [16] proposed a neural network system that predicts the optimal points to make market decision and output buy or sell rather than forecasting the exact numerical price value of a stock. Moreover, this work make predictions upon the market index of different stock exchanges, for example Dow Jones Index, S&P 500, instead of a single stock’s closing price as in [15]. By using only the historical price data as input to train the neural network, their experiments generate significantly greater return comparing to the well known random walk model.

In addition to different evaluation standards used in the system, input selection is also one of the important factors that could significantly impact the performance of neural network prediction. For instance, [17] suggested that besides historical price data, there is a strong relationship between future stock price and trading volume. Experiments were conducted to discover the impact of volume on the future stock price in short-term, mid-term, and long-term time frames. Using market index as evaluation criteria, the results have shown that the long-term predictions with presence of trading volume generate the best performance out of all three time frame horizons.

In these earlier studies, different combinations of forecasting targets and input features are tested and results have shown that neural network could generate considerably good predictive results, however due to the limited training data and computing power available back then, normally a small and shallow neural network was used to model the market data, and the model was usually generalized on a small dataset which makes the result less promising. In addition, only numerical market data were used to construct
the feature vectors for neural networks, such as historical prices, trading volumes, etc, in order to predict future stock yields and market returns. The behaviour of such market data appears to be extremely volatile because of the nature of the market and the existence of large amount of noise mixed in the original data, therefore the performance on the system with use of such data is limited.

More recently, in the community of natural language processing (NLP), many methods have been proposed to explore additional information (mainly online text data) for stock forecasting, such as financial news and reports, twitters sentiments and microblogs. For instances, [24] and [27] extracted NLP features from financial news that were collected from the internet; [24] proposed to use semantic frame parsers to generalize from sentences to scenarios to detect the (positive or negative) roles of specific companies, where support vector machines with tree kernels were used as predictive models, whereas [27] proposed to use various lexical and syntactic constraints to extract event features from financial news for stock forecasting, where they have investigated both linear classifiers and deep neural networks as predictive models. [18] conducted research that extracted features from the 8k reports which were labeled with important financial events associated with the company, both events and market index informations were used as features to train the Non-negative matrix factorization (NMF) model in their work. On the other hand, [25] proposed a continuous Dirichlet Process Mixture model to learn the topic set of Twitter messages, and the sentiment of the Twitter topics were used to make predictions on the stock market. [26] built a Semantic Stock Network base on Twitter messages that models the semantic relationships among stocks; a vector autoregression model was then introduced to predict the sentiment of the relationship as well as the stock price movement. In addition, [28] leveraged the stock microblogs, which is a collection of tweets that has been filtered to be just related to expert investor and stocks,
and they defined the level of expertise of such microblogs in order to predict the stock price; both supervised and unsupervised learning model were tested in they work.

These proposed methods with NLP features have shown that textual data could make considerable impact on the predictive performance of stock forecasting tasks, however few of these works applies the popular neural network model to learn the features, thus in this thesis, we propose to use neural network as our predictive model to learn from the financial news. Moreover, with the availability of fast GPU and large training dataset, we would like to discovery how bigger datasets and deeper network architecture will impact the predictive performance of the neural network approach. In addition, we propose an approach to leverage both historical stock features and NLP features to establish a machine learning system for stock price forecasting.

1.3 Goals of this thesis

A typical stock market predication method will likely predict the future value of a stock, in particular, predicting the exact price value of it. Since neural network has been proven to be good at classification problem rather then regression problem. We decided to predict the moving direction the stock instead of the price value, so the DNN will have outputs with the form of 1(price raise) or -1(price drop).

With the setup of this form of output, our main focus in this thesis is to construct a model that will minimize the classification error, and generate as much prediction as possible base on available data. We first use historical stock market data to construct input features in our model and explore the impact of larger amount of training dataset as well as bigger and deeper neural network architecture comparing to previous work. Our experiments also benefits from GPU which significantly accelerates the training
process. In addition to the historical data, we then propose to use the recent word embedding methods \cite{30, 41, 42} to select features from online financial news corpora, and employ deep neural networks (DNNs) to predict the future stock movements based on the extracted features. Our experiments will discover the predictive performance of different combinations of features, and the goal is to find out the importance of different features. In fact, our experimental result shows that the features derived from financial news are very useful and they can significantly improve the predictive accuracy over the baseline system that only relies on the historical price information. Moreover, we propose a method called correlation matrix to extract the potential relationships among stocks which is aiming for expand the prediction results produced by DNN.

### 1.4 Organizations of this thesis

The rest of the thesis is organized as follow. Chapter 2 will introduce the predictive model we are using in this thesis: deep neural network, including the construction of such model as well as the back-propagation training algorithm we use to train the model. Chapter 3 first briefly introduces the concept of Natural Language Processing, then explains the word embedding method in detail; we apply word embedding method in our work to search for similar keywords from training corpora and we construct the feature vectors from those keywords. Chapter 4 gives a background introduction of financial modelling and some traditional stock price analysis methods, such as technical analysis and fundamental analysis, which motivates the design of feature vectors in this work. In Chapter 5 we described how to construct our baseline system which is using the features purely extracted from the historical stock data. Then in chapter 6, we focus on how to use word embedding method to extract different NLP features from financial
news. Chapter 7 will present the correlation matrix method that take the output from neural network and propagation them into a larger domain of stock collections in order to further expand the prediction results. Then we show the experiment setups in chapter 8 and discuss the experimental results as well as the discoveries derived from those results. Finally in chapter 9 we conclude our contributions as well as future steps.
Chapter 2

Deep Neural Networks

In this chapter, we will introduce the primary predictive model used in this thesis, deep neural network. Which takes the input of features extracted from both historical price information and online financial news to predict the stock price movements in the future (either up or down).

Neural networks have been proven to be powerful predictive models. In general, it is a model with fixed number of basis functions in which there are adaptive parameters. The parameters are then adjusted during the process of training. There are several different variants in terms of the structures of neural networks. For examples, the conventional structured feed-forward network [38] is a model contains multiple fully connected layers of perceptrons. Convolutional neural network (CNN) [39] is the network with convolution and pooling structure that has been shown to be powerful in image processing tasks such as object detections. Recurrent neural network (RNN) [32] is a network with the structure that besides the full connections between adjacent layers, there is also connections of the hidden unit itself, which forms a directed circle. This form gives RNN the ability to store temporary states information and make it applicable...
Figure 2.1: An simple example of neural network with a single hidden layer. Each circle represents a computing unit in the network. Each arrow represents the connection between two units. The direction of arrow indicates that the input information is propagated from input layer at the bottom to the output layer at the top.

for the problems that involve sequence processing with arbitrary input size, for instance, RNNs have been proven to generate strong results in language modeling and machine translate. The structure of neural network used in this thesis is a conventional multi-layer perceptron with many hidden layers. We will explain the detailed information of its structure in the following section.

2.1 Structure of feedforward neural network

The basic idea of the artificial neural network is inspired by human brain’s structure, where there are billions of neurons that are connected by synapses to form the biological neural network. Each neuron is a computing unit that process the incoming information and pass the results to next connected neurons. Base on this structure, an artificial neural network is also designed to be constructed by a set of computing units [43]. The
computing units are further organized into different layers in the neural network. The two layers at the end of this layered structure corresponding to the network’s inputs and outputs. Usually, the bottom layer is called input layer, the layer at very top is called output layer and all the layers in between are called hidden layers, in addition, the unit belongs to a hidden layer is then named as a hidden unit. In the neural network, each layer is fully connected to each other, meaning there are connections between each pair of units in adjacent layers and there is no connection between two units that belong to the same layer. An example of a neural network with a single hidden layer is shown in figure 2.1. When propagating information, the input layer will take a series of inputs and feed them to a set of hidden units that are connected to them through the connections between inputs and the hidden units. The connections are adjustable parameters in this model, and each connection is responsible for controlling the amount of effect the corresponding input will pass to the connected hidden unit. Once the inputs reach the hidden unit, they will be processed or thresholded by a non-linear activation function in the unit to generate an output. Since the layers are fully connected to each other, the outputs of one layer of hidden units will be further passed to the next layer of hidden units and serve as a set of new inputs to the next layer. Therefore, the original inputs act like a series of signals that will be propagation through the whole network until it reaches the output layer and generate the final outputs of the network.

We will now describe the main components of the neural network using more precise mathematical notations, which includes: network’s input, feed-forward network function, activation function and network’s output.
2.1.1 Network’s Input

The input of a neural network is a vector with fixed size, for example, $\mathbf{x} = [x_1, x_2, \ldots, x_N]$, where $\mathbf{x}$ is the input vector and $N$ is the size of input vector. In figure 2.1, $N$ also represents the number of units in the input layer.

2.1.2 Feed-forward network function

The feed-forward function is the basic of neural network model and can be defined as a series of functional transformations. First, assume the hidden layer connected to the input layer has $M$ hidden units. For each hidden unit $j$, the input vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ will be transformed using the following linear combination:

$$a_j = \sum_{i=1}^{N} w_{ji}^1 x_i + w_{j0}^1$$

where $j$ belongs to $[1, M]$. $w$ are the adjustable parameters that attached to each connection between the input unit and the hidden unit, which is also called \textit{weights}. The superscript 1 of $w$ indicates that the weights are corresponding to the connections going toward the units in the first hidden layer. $w_{ji}$ denotes the weights between input unit $i$ and hidden unit $j$. In addition, $w_{j0}$ represents the \textit{bias} similar to the bias term in the basis function in linear regression. This bias can be simply represented as a connection between the hidden unit and an additional input unit whose value is always 1. The result of this transformation is stored in $a_j$ which is known as \textit{activation}.

Each one of the hidden units in the first hidden layer will receive an activation $a_j$ as input, and further transform them using the nonlinear \textit{activation function} $g(x)$:

$$z_j = g(a_j)$$
$z_j$ is the output of the hidden unit. It should be mention that $g(x)$ is a function that is differentiable, this is a really important property for the training stage of neural network so that we can backpropagate the error.

Once the outputs flow through the first hidden layer and transform into output $z_j$, $z_j$ becomes the input for the next hidden layer or output layer in the case there is no more hidden layer. In this example, we assume there is only one hidden layer for simplicity. Therefore $z_j$ will be fed to the output layer, and we further assume there are $K$ units in the output layer. The outputs of the previous hidden layer form a new vector $\mathbf{z} = \{z_1, z_2, ..., z_M\}$ which will be fed to the output layer as the input vector. This input vector is again linearly combined to give the activation of output units:

$$a_k = \sum_{j=1}^{M} w_{kj}^2 z_j + w_{k0}^2$$

where $k$ belongs to $[1, K]$. The superscript of $w^2$ indicates it is the weight corresponding to the second layer of the connections in the neural network. Each $a_k$ will then transformed using activation functions in the output unit to finally generate the output of the neural network. The activation function in output unit is chosen depends on the problems which neural networks are learning. In addition, it is usually different from the ones used in the hidden unit. We will discuss the detail of output function shortly.

Now we can combine these previous transformations in each layer and stack them together to give the mathematical definition of a single hidden layer neural network:

$$NN(\mathbf{x}, \mathbf{W}) = f\left(\sum_{j=1}^{M} w_{kj}^2 g\left(\sum_{i=1}^{n} w_{ji}^1 x_i + w_{j0}^1 + w_{k0}^2\right)\right)$$

where $f$ is the output function. To make this form simpler, we can rewrite each layer’s
linear combinations in a more compacted way. As we described earlier, the bias term in each layer’s linear combination can be simply represented as an additional connection coming from an input unit that is always equal to one. Therefore we can add an extra element $x_0$ into each layer’s input vector $\mathbf{x}$ and set the value of $x_0$ to 1. Then the bias term can be combined into the weight matrix in the corresponding layer. For example, the linear combination of the first layer will be rewritten as:

$$a_j = \sum_{i=0}^{N} w_{ji}^1 x_i$$

Therefore, if we move each layer’s bias term into their corresponding weights matrix, the neural network’s function becomes:

$$NN(\mathbf{x}, \mathbf{W}) = f\left(\sum_{j=0}^{M} w_{kj}^2 g\left(\sum_{i=0}^{n} w_{ji}^1 x_i\right)\right)$$

In this function, $\mathbf{W}$ is the vector that groups together all the weights and bias in each layer, each element in $\mathbf{W}$ is a matrix contains the weights between two adjacent layers, and the size of the weight matrix depends on the number of units in the connected layers. From this mathematic form, we can see that the neural network model is a simply a non-linear function that maps a set of input variables $\mathbf{x}$ to a set of output variables. The forward propagation of input information through the whole network can be represented as the evaluation of above function. In addition, $\mathbf{W}$ can also be viewed as a set of adaptive parameters that controls the output of this function. Therefore, we call $\mathbf{W}$ the parameter of the neural network, and a training algorithm for a network is responsible for setting the values in $\mathbf{W}$ correctly such that the neural network will make correct predictions.
2.1.3 Activation function

The activation function is one of the most important component in the neural network, which provides the non-linearity to the model. The choice of activation function can considerably impact the performance of the neural network. First of all, the activation function should be differentiable so that the overall neural network function is differentiable and make it possible to run error back propagation algorithm, which will be discuss in next section. Second, the activation function should be chosen as a non-linear function which gives the neural network the ability to generalize on a large range of functions. Figure 2.2 shows examples of two widely used activation functions.

One of the popular choices of activation function is sigmoid function. It is a “s” shape function with the mathematic form:

\[
sigmoid(a) = \frac{1}{1 + e^{-a}}
\]

this form takes the real value input and transforms it into the range between 0 and 1. The advantage of using sigmoid as the activation function is that it mimic the firing rate of the human neuron, the output of 0 act as a not firing states in neuron and output of 1 act as the firing states. However, one of the problems of using sigmoid activation function is that when the input is too small or too big, the gradient of this function approaches zero and the signals for error back propagation will vanish.

Another choice of activation function is rectified linear function or rectifier which takes the form:

\[
ReLU(a) = max(0, a)
\]
Figure 2.2: Two examples of activation functions.

It simply threshold the input at 0, output the input if it is greater than 0 and output 0 otherwise. The rectifier activation function becomes very popular in the recently few years because of its simplicity which improves the training speed of the neural network in a great deal. It also produces great results since the rectifier unit does not have the vanishing gradient problem.
2.1.4 Network’s output

The setup of neural network’s output layer depends on the problem it is dealing with. If the output has \( k \) dimensions, where \( k > 1 \), the network can be used for \( k \)-class classification problem. If the network’s output only has one dimension, then this kind of network can be used for regression problem. Moreover, the activation function in output layer will also depend on the targeting problem. If the network is learning a set of data for standard linear regression problem, the output activation function is simply the identity of the incoming activation. If the neural network is used as a classifier that models the probability distribution over the \( k \) possible output classes, the output activation function is usually set to be the softmax function. For example, if the network is a classifier for a \( k \)-class classification problem, then the output layer will have \( k \) units, whose output activation function looks like:

\[
softmax(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{k} e^{a_j}}
\]

where \( a_i \) denotes the activation of \( i \)-th output unit. For the input vector \( \mathbf{a} = \{a_1, a_2, ..., a_k\} \), the result of output layer after the softmax transformation is a vector of non-negative real numbers that sum up to one. Each dimension in the result vector becomes a conditional probability of the corresponding class, ie. \( y_i = P(y = i|\mathbf{x}) \), where \( \mathbf{x} \) is the input vector and \( y_i \) is the \( i \)-th dimension in output vector.

In this thesis, we use a deep neural network with more than one hidden layer. Each hidden unit uses the rectified linear activation function, i.e., \( f(x) = \max(0, x) \), to compute the outputs in each hidden layer, which are in turn fed to the next layer as inputs. Since we are predicting the moving directions of the stock, the output layer will have only two units, standing for two classes of \textit{stock-up} and \textit{stock-down}. Within each output unit, the
softmax function is chosen to compute posterior probabilities for these two classes. Different depth of the network, as well as the size of each hidden layer, are tested in order to get the best predictive performance, details of the experiment and tuning process will be explained in chapter 8.

2.2 Training DNN

In order to train a neural network model, we use a method that retrieves the gradient information from network’s output error, to minimize a loss function. In general, a neural work training algorithm is an iterative function that repeatedly feed a set training data into the network, compares the actual output with expected output and compute the gradient information from the difference; and then adjust the network’s parameter according to the gradient information in order to minimize the loss function.

2.2.1 Loss function

Before training neural network model, we need to define a loss function, or objective function, that measures the progress of the training procedure. For example, a loss function loss(y, y'), takes the input y which is network’s output vector contains predictive value for each output unit and y' is the vector stores actual correct values of the corresponding input samples. Such loss function will output the loss of predicting y when the correct output should be y'. Therefore, a neural network training algorithm is subject to minimize the value output by loss function given a set of input training samples. A positive real value score will be assigned to each network output by loss function according to the correct value, in the case the output is exactly the same as
expected correct value, the loss function will assign zero, this is why we also call the
expect output value as target.

Similar with output activation function, the choice of loss function is also depends on the
type of problem the network is training for. If the neural network is use for regression
problem, one of the common choice is the sum-of-squares function with the form:

\[ loss(y, y') = \frac{1}{2} \sum_{i=1}^{N} (y_i - y_i')^2 \]

where \( N \) is size of the output vector, \( y_i \) and \( y_i' \) are the matched network output and target
output for \( i \)-th unit. This loss function basically sum up the absolute difference between
actual output and expect output. For the networks that dealing with classification
problem, we use cross entropy function with the form:

\[ loss(y, y') = - \sum_{i=1}^{N} y_i' \log(y_i) \]

where \( N \) is the total number of available classes. \( y' \) is the target probability distribution
of the input sample over \( N \) classes, it is a one-hot vector with the dimension corresponds
to correct class set to 1 and every other dimensions set to 0. \( y \) is the softmax output
vector with each dimension set to the predictive probability on each class. The value of
this loss function represents how much the predictive distribution \( y \) is differ from the
expected distribution \( y' \).

To give a particular example, for the neural network established in our thesis, the output
layer has two units, each of which generates a probability of the associated class by
softmax function, so \( N = 2 \). Assuming the output vector \( y = \{0.1, 0.9\} \), which indicates
the input training sample has 10% chance belongs to the class of “stock-up”, and 90%
chance belongs to the class of “stock-down”. Also assume the correct class of the input is “stock-up”, then the expected target output will be $y' = \{1.0, 0.0\}$. So the loss function is $\text{loss}(y, y') = -\sum_{i=1}^{2} y'_i \log(y_i)$. Because $y$ is the output of the network, $y = NN(x, W)$. Therefore the training algorithm’s goal is to adjust the weight matrix $W$ subject to minimize $\text{loss}_{\text{cross-entropy}}$.

### 2.2.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a common training algorithm for neural network. In general, SGD algorithm takes the following inputs: a function $f(x, \theta)$, where $\theta$ is the adjustable parameter of $f$; a training dataset includes the input samples $X = x_1, x_2, ..., x_n$ and the corresponding outputs, or labels, $Y = y_1, y_2, ..., y_n$; and a loss function. In the case of training neural network, the input function $f$ is the network’s feed forward function and the adjustable parameters is the matrix of weights $W$. The goal of SGD is to repeatedly set the parameter $\theta$ such that the value of loss function will be minimized until the error of $f$ converge.

In each iteration, the algorithm first randomly choose a sample input output pair $\{x_i, y_i\}$ from the training dataset, and feed $x_i$ to function $f$. Then it computes the value of loss function by $\text{loss}(f(x_i, \theta), y_i)$, and get the gradient of the loss function with respect to the parameter $\theta$. Finally, it updates the parameter using the gradient information in the following way:

$$\theta = \theta - \eta \nabla \text{loss}(f(x_i, \theta), y_i)$$

where $\nabla \text{loss}(f(x_i, \theta), y_i)$ denotes the gradient of loss function, and $\eta$ is a term called learning rate that use to scale the amount of gradient to be use to move the parameter. The algorithm will constantly repeat the above iterations until the change of parameter
\( \theta \) does not considerably reduce the value of loss function anymore, which we called converge.

In the last step of each iteration, the value of learning rate represents the pace of gradient descent, it is an important attribute in the algorithm that should be set carefully because if we set it too big, the pace of adjusting \( \theta \) will be too large that the algorithm will bounce around a local minimum of the loss function and never converge. This value also should not be set to be too small since a small descent pace will slow down the converge speed. Some methods are applied to dynamically set the learning rate which give the algorithm better converge value. One method is to set the learning rate as a function of time such that \( \eta = 0.9^i \eta \), where \( i \) is the number of iterations. This method constantly reduces the value of learning rate with a ratio in every iteration so that at the beginning of the algorithm the learning rate is big enough for the algorithm to effectively reduce the value of loss function. With the increment of the iterations, the learning rate dropped for the algorithm to move with small paces around the local minimum to find the best converge point. Another method, which is applied in our experiment, is to use a validation dataset that contains input and output pairs have not appear in the training dataset and add one more validation step in each iteration of SGD. Every time after the algorithm updates the parameter \( \theta \) to \( \theta' \), we first feed the inputs from validation dataset to the network with updated parameter \( \theta' \), and remember the validation error between networks output and corresponding validation label using loss function. Then, we compare the validation error in the current iteration and previous iteration to see if we need to adjust the learning rate. In the case that the validation error is higher than the one in the previous iteration, which means the learning rate is too high so the algorithm made too much change on \( \theta \) and missed the local minimum, we reduce the learning rate with certain ratio and revert the parameter back to \( \theta \). The whole
iteration is then rerun with updated learning rate. In the case that validation error has reduced comparing to the previous iteration, the learning rate remains unchanged and the algorithm keeps moving forward.

One problem with the above SGD algorithm is that when computing the error in each iteration, we only use a single training sample to estimate the overall loss across the whole training dataset, which will be ambiguous. In addition, a real life training dataset may contain many noise samples; hence, in an iteration using noise as input sample, the network’s weights will be moved to a direction that is far away from the correct minimum point. To address this problem, a common workaround is to use $N > 1$ training samples instead of using only one in each iteration, which we called the mini-batch SGD. In mini-batch SGD, we feed a set of $N$ training samples to the function $f$ in every iteration, and then use the average gradient over the $N$ samples’ loss function to update $\theta$:

$$\theta = \theta - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla \text{loss}(f(x_i, \theta), y_i)$$

In this way, the SGD is estimating the error’s gradient over the $N$ samples. The size of the mini batch can change depends on different network setups. Using a mini batch with a smaller size will accelerate the training speech, by providing a larger gradient in each iteration, but the loss value will fluctuate heavily during the training process. While a bigger size will provide a better estimation of the overall gradient across the training dataset and, therefore, has a smoother descent of gradient. Moreover, a system with mini batch method could also benefit from the parallel computing provided by modern GPU, hence improve the training efficiency.

In terms of applying SGD in neural network training, the main challenge is to compute the gradient of network’s loss function with respect to each layer’s weights. Usually, $W$
is a big matrix in a deep neural network architecture, therefore computing the derivative of the loss function with respect to $\mathbf{W}$ can be cumbersome. [19] provides an easy solution called backpropagation algorithm to address this problem.

### 2.2.3 Error Backpropagation

*Error backpropagation algorithm* is an efficient technique to evaluate the gradient of a loss function for a feed-forward neural network. The goal is to find out the derivative of loss function with respect to the weight matrix, i.e. $\frac{\partial L}{\partial \mathbf{W}}$, with $L$ denotes the loss function that takes the network’s output and the expected target as input. Recall that the network’s output is calculated by the feed-forward function which has a nested structure with many layers of linear combinations:

$$NN(\mathbf{x}, \mathbf{W}) = f\left(\sum_{j=0}^{M} w^2_{kj} g\left(\sum_{i=0}^{n} w^1_{ji} x_i\right)\right)$$

The difficulty lies on the evaluation of the derivative with respect to the most inner weight matrix that is “far away” from the loss function, because computing the derivative with respect to those matrices will require unfolding the entire nested structure. Therefore, the deeper the neural network is, the harder to unfold the expression. The error backpropagation algorithm simplifies the process of evaluating loss function’s gradient by passing the local messages to propagate the network’s error backward through the network during the evaluation of the derivative of the loss function.

For simplicity, we will explain the backpropagation algorithm using the example of training a simple neural network with a single hidden layer, and a single sample is used in each iteration of SGD. Hence the hardest part is to compute the derivative of loss function with respect to the weight matrix corresponds to the connections between input
layer and hidden layer. We now consider the derivative with respect to a single weight $w_{ji}$, such that $w_{ji}$ represents the weight associated with the connection from input unit $i$ to hidden unit $j$. To find out the derivative $\frac{\partial L}{\partial w_{ji}}$, we observe that the way loss function $L$ changes along with $w_{ji}$ is through the activation of hidden layer:

$$a_j = \sum_i w_{ji} x_i$$  \hspace{1cm} (2.1)

where $x_i$ is the inputs of the neural network. Also note the bias term has been absorbed into the weight matrix by setting a constant input 1, so we do not need to take care of it separately. This summed input of hidden unit $j$ is then transformed by the activation function $g$ to further provide input to output unit:

$$z_j = g(a_j)$$  \hspace{1cm} (2.2)

Therefore we can apply the chain rule for computing the derivative of the composition of two or more functions on the derivative of loss function with respect to $w_{ji}$:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$  \hspace{1cm} (2.3)

On the right hand side of this equation, we can rewrite $\frac{\partial a_j}{\partial w_{ji}}$ using equation (2.1) so that:

$$\frac{\partial a_j}{\partial w_{ji}} = x_i$$  \hspace{1cm} (2.4)

In addition, we denote the term:

$$\delta_j \equiv \frac{\partial L}{\partial a_j}$$  \hspace{1cm} (2.5)
where $\delta$ is commonly referred as *error* since this is the term contains the error gradient information being propagated through the network during the evaluation of the derivative of loss function. Here in particular, $\delta_j$ represents the error corresponds to the weights connecting input and hidden unit.

After combining equations (2.4) and (2.5), equation (2.3) can be rewrite as:

$$\frac{\partial L}{\partial w_{ji}} = \delta_j x_i$$

(2.6)

The simplified equation evaluates the derivative of loss function by multiplying two terms at two ends of the connection, respectively, associated with weight $w_{ji}$, where $x_i$ is the value provided by the input end of the connection and $\delta_j$ is the value correspond to the unit at the output end of the connection. Therefore the next step of evaluating the required derivative is to compute the value of $\delta_j$ using the information in hidden layer and output layer of the network, and then insert the result back to equation (2.6).

Again, in equation (2.5), we observe that the way $L$ changes with $a_j$ is only through the activation of output layer $a_k$. Therefore we can also apply the chain rule on (2.5):

$$\delta_j = \frac{\partial L}{\partial a_j} = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

(2.7)

note here we need to sum up $k$ activations of output unit. For the reason that in the feed forward network function, the output transformation of hidden unit $j$:

$$z_j = g(a_j)$$

(2.8)

has been propagated to all $k$ units in the output layer because of the fully connected structure.
With the similar simplification we did before, denoting \( \delta_k \equiv \frac{\partial L}{\partial a_k} \) and evaluate the derivative of \( \partial a_k \) over \( \partial a_j \) as \( \sum_k \frac{\partial a_k}{\partial a_j} = g'(a_j) \sum_k w_{kj} \), we finally get the back propagation formula:

\[
\delta_j = g'(a_j) \sum_k w_{kj} \delta_k
\]  

(2.9)

which evolves evaluation of two simpler derivatives, \( g'(a_j) \) is just the derivative of the activation function used for all hidden units; and \( \delta_k \) is the derivative of loss function with respect to the output units’ activation, this term depends on the choice of output activation function and loss function. This equation shows that in a network with a single layer of hidden units, the value of \( \delta \) of a hidden unit can be computed by propagating back the \( \delta \)'s of all the proceeding output units connected to that hidden unit. The result of \( \delta_j \) can be plug back into equation (2.6) to get the required gradient. Such pipelining of the derivatives with respect to a single layer of a matrix can be interpreted as a local computation that pass through the entire network. For a network with deeper structure, i.e. more than one hidden layer, we can recursively apply equation (2.9) to evaluate the \( \delta \) values of each hidden layer.

The overall procedure of error back propagation is illustrated in figure 2.3. To sum up, the back propagation algorithm first takes the input vector and feed it through the neural network to get the network output as well as the activation of all hidden units. Then it computes the derivative of the loss function with respect to the output units’ activation \( \delta_k \). In the backpropagate stage, it backpropagates the \( \delta \)'s, starting from \( \delta_k \), to get all errors for every hidden unit in the network. At last, the algorithm inserts \( \delta \)'s back to equation (2.6) to evaluate the required gradients.
Figure 2.3: Overall procedure of error back propagation algorithm. Blue arrow indicates the direction of feed forward networks flow from input layer to output layer. Red arrows indicates the direction of backward error propagation.
Chapter 3

Word Embedding

In this chapter, we will discuss the concept of word embedding. Word embedding is a technique that is derived from language modeling in the natural language processing (NLP), it generates a distributed representation of words in a vocabulary. Such representation maps the word into a vector space, and each of them is then represented by a vector of real numbers. There are several advantages of this word embedding technique comparing to the traditional bag-of-word representation of words. For example, the vector representation of words eliminated the common difficulty in NLP tasks which is known as the curse of dimensionality. Another attracting property of this method is that the vector representation preserves the similarity of the words in the natural language, meaning the similar words should have similar embeddings. As we will show in chapter 6, the property of similarity can help us find useful features from the financial news corpus. There are several different methods of generating such word embeddings. In this thesis, we are interested in the popular Skip-gram and Continuous bag-of-words models, which are simplified from neural net language modeling.
3.1 Language modeling

A language model is a model that estimates the probability distribution of words in
the vocabulary or a joint probability of a sentence. For instance, one task for lan-
guage modelling is to generate the probability of a sentence or a sequence of \( m \) words
\((w_1, w_2, ..., w_m)\), which can be written as the following term produced by chain rule:

\[
P(w_1, w_2, ..., w_m) = \prod_i P(w_i|w_1, w_2, ..., w_{i-1})
\]

where \( P(w_i|w_1, w_2, ..., w_{i-1}) \) is the conditional probability of \( w_i \) given the previous word
sequence is \((w_1, w_2, ..., w_{i-1})\). In addition, a language model could be used to predict the
missing word in a sentence. Given a sentence “The cat sat on the mat.”, if we take alway
any word from the sentence, for instance, the last word “mat”, the language model can
predict the missing word by taking the remaining sequence “The cat sat on the” and
assign a probability to each of the words in the vocabulary.

The ability to estimate relative likelihood of sentence makes language modeling very
useful in natural language processing tasks. There are many applications base on it,
ranging from spelling correction, speech recognition, machine translation to part-of-
speech tagging and parsing. For instance, in speech recognition, the system needs to
translate a segment of acoustic input into a word sequence. The probability distribution
provided by a language model enables the speech recognition system to distinguish
the words or phrases that have similar pronunciation. Given two phrases with totally
different meaning: “I saw a van” and “eyes awe of an”, which are pronounced almost
the same, it is hard to decide which one is the correct translation using only the acoustic
model. With the aim of the language model, it would be shown that the probability
of the phrase ”I saw a van” appears in English is much higher than the probability of
"eyes awe of an". The generated probability of this two phrases can then provide useful
evidence to help the system to make a correct decision.

There are several types of language models, different language implements the term of
conditional probability $P(w_i|w_1, w_2, ..., w_{i-1})$ differently. An n-gram model is one of the
most traditional language model which is often used as a benchmark model for other
techniques. In this model, it simply counts the frequency of word or word sequence to
obtain the conditional probability. Moreover, it applies the Markov assumption which
assumes that the probability of a word only depends on the previous $n$ words:

$$P(w_1, w_2, ..., w_m) = \prod_i P(w_i|w_{i-n}, ..., w_{i-1})$$

in which the conditional probability of $w_i$ is:

$$P(w_i|w_{i-n}, ..., w_{i-1}) = \frac{\text{count}(w_{i-n}, ..., w_{i-1}, w_i)}{\text{count}(w_{i-n}, ..., w_{i-1})}$$

Such approximation is usually “smoothed” by adding 1 to the nominator to handle the
situation that the unseen word sequences appear in the test dataset, which causes the
probability becomes zero.

The simplest case of n-gram model is the unigram, where $n = 1$ and the probability of
a word is just the frequency of itself:

$$P(w_1, w_2, ..., w_m) = \prod_i P(w_i)$$

In addition, the n-gram model with $n = 2$ and $n = 3$ are called bigram and trigram
respectively.
Even though such n-gram language model has been found to be useful in many NLP applications as we mentioned before, it suffers from a fundamental problem: *curse of dimensionality*. It is a phenomenon that with the increase of the dimensionality of data, the volume of space increases so fast that the data become really sparse within the vector space. In a statical language model such as n-gram model, as the training corpus of the model becomes larger, the vocabulary of unique words appear in the corpus becomes bigger, which leads to a task of modelling the probability of exponentially many of word sequence, therefore the word sequence in the training corpus becomes sparse in the space of vocabulary.

A *neural net language model* (NNLM) is proposed in [9] to alleviate such problem. In a neural net language model, the probabilities are predicted using a neural network instead of simply counting the frequency. As we discussed in the previous chapter, a neural network can be used as a classifier that assigns the input sample a probability of belonging to a specific output class. In an NNLM, the typical task is training a neural network to generate the probability distribution over the words in a vocabulary: \( P(w_i|\text{context}) \), where context could be the previous \( n \) words before \( w_i \), in which the goal is predicting the upcoming word; or context could be previous \( n \) words concatenated with following \( n \) words of the missing word, in which the missing word in the middle of a word sequence is predicted. The output of NNLM is, therefore, a \( V \)-dimension vector, where \( V \) is the size of the vocabulary and \( i \)-th dimension of the vector correspond to \( i \)-th word in the vocabulary. Each dimension of such vector then contains a probability assigned to the corresponding word.

In a neural network trained for NNLM, an *embedded layer* is added before the input layer to map the word in vocabulary into a distributed representation. Each word in the
vocabulary is translated into a $d$-dimension vector, for example:

$$C(\text{“cat”}) = (0.2, -0.4, 0.7, \ldots) \text{ or } C(\text{“mat”}) = (0.0, 0.6, -0.1, \ldots)$$

where $C(\cdot)$ is a function represented by a $|V| \times d$ matrix and each row of C is a vector associated to a unique word. A sequence of such word vector is then either concatenated together or summed up to an input vector for the network.

Such distributed representation has several remarkable properties. First of all, comparing to the traditional one-hot representation of word feature (representing each word as a unique dimension in a $V$ dimensional space), the distributed representation embeds each word into a much smaller $d$ dimensional space which makes the resulting vector much more dense, therefore reduces the impact of the fundamental curse of dimensionality problem. Second of all, such representation preserves the similarity information between words; the words with similar meaning are close to each other in terms of the cosine distance between the two corresponding vectors. This property also helps handle the problem of appearance of unseen word sequence in the test dataset; for example, if we change the word sequence “The wall is red” to “The wall is blue”, since similar word will have similar vectors, the representation of these two sentence will be similar even if the sequence “The wall is blue” has never appear in the training data before. Moreover, this distributed representation captures the semantic meanings between words. In contract with similar words having similar vector, antonyms of word tend to have dissimilar vectors and such similarity and dissimilarity can be expressed in terms of vector arithmetic, for instance:

$$C(\text{“queen”}) - C(\text{“women”}) + C(\text{“man”}) \approx C(\text{“king”})$$
Because of these advantage of such distributed representation of words, besides using
the neural net language model to produce actual probability, it is also common to use
the distributed representation encoded in the network’s embedded layers as the repre-
sentations of words and solve other NLP problems.

3.2 Learning distributed representation of words

In many tasks, the embedded layer is pre-trained in order to obtain a reliable embedding
function and then plugin to the neural net language model. [1] proposed two models
that learn a high-quality embedding function base on a large dataset in an efficient way.
As shown in figure 3.1, the architecture of these two models is similar to the feedforward
neural network with a layered structure.

The first model is called continuous bag-of-words model (CBOW). Roughly speaking, this
model takes a sequence of words with a certain window size and predicts the middle word
of the sequence. For example, as shown on the left hand side of figure 3.1, a sequence
of 5 words \( \{x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}\} \) is use to construct the input and output vectors
for the model. The surrounding words \( \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\} \), which is also called the
context of \( x_t \), is use as input and the middle word \( w_t \) is use as the output of the model.
Each of these input words is represented as a one-hot \( V \)-dimension vector where \( V \) is
the size of the vocabulary. The weight matrix between the input layer and hidden layer
is a \( V \times d \) matrix that is shared by every input words. Since each of the input words is
represented by a vector with all dimensions set to zeros except the one dimension, which
corresponds to the position of the input word in the vocabulary, set to one; the linear
transformation from an input word to hidden layer is simply a linear projection. The
projection of each input word is then stacked together and averaged as the input vector
Figure 3.1: Architecture of two models for learning high quality word embedding proposed in [1]. Both models use the word sequence \( \{x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}\} \) as an example. CBOW takes \( \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\} \) as input and predict the middle word \( x_t \). Whereas skip-gram takes the middle word \( w_t \) as input and predict the surrounding context \( \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\} \).

to the hidden layer. The hidden layer contains \( d \) hidden units, but different from the normal neural network’s hidden unit which contains a non-linear activation function, the hidden unit in this model does not have any activation function. Therefore, the output vector of hidden layer is simply as same as its input vector. The output layer contains \( V \) output unit with softmax output function that will generate a probability distribution over the words in the vocabulary. Therefore, the training objective of this model is maximizing the log probability of word \( x_t \):

\[
\log p(x_t|\text{context})
\]

where \( \text{context} \) is the surrounding words \( \{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\} \), and \( p(x_t|\text{context}) \) is the
probability of middle word $x_t$ given the surrounding words. The name continuous bag-of-word comes from the conventional bag-of-word model which also does not consider the order of words within a sequence, and instead of $I$ of $k$ representation in bag-of-word, this model embeds the continuous vector representations in the embedding function.

The second model is called *continuous skip-gram* model, shown on the right hand side of figure 3.1, which is similar to CBOW but reverses the direction of prediction. Instead of predicting the missing word in the middle of a word sequence, the skip-gram model takes single word $x_t$ as input and predicts the previous and next words in a certain distance. In the example of figure 3.1, the skip-gram model predict previous and next 2 words $\{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\}$. The input layer takes an one-hot representation of word $w_t$ and, similar with CBOW, project it to the a hidden layer consist of hidden nodes without any activation function. The output layer consist of 4 output vector each of which is the one-hot encoded word in the vocabulary with half of them being the words previous to $x_t$ and half of them following $x_t$ in a word sequence. Each output layer has an associated $d \times V$ weight matrix. The training objective of skip-gram model is maximize the log probability:

$$\sum_{-c \leq i \leq c, i \neq 0} \log p(x_{t+i}|x_t)$$

where $c$ is the size of context window, and $c = 2$ in the example of figure 3.1; $p(x_{t+i}|x_t)$ is the probability of $x_{t+i}$ being the $i$-th word before $x_t$ if $i < 0$ or after $x_t$ if $i > 0$.

In both of these two models, the weight matrix between the input layer and the hidden layer is the target we looking to learn since it contains all the embedding information for the vocabulary. The training procedure is similar of what we have discussed in feed forward neural network. First, the weight matrices are initialized using random real
values. Then the word sequences are sampled from the training corpus to construct input and expected output vectors, the input vectors are fed to the network to generate actual predictive outputs. Finally, base on the predictive outputs and expected outputs, the weights are adjusted using SGD and backpropagation algorithm.

Both CBOW and skip-gram model learn distributed representation of words efficiently by removing the computation complexity generated by hidden layers. In addition to these two models, [30] proposed an extension of the skip-gram model to not only extends the representation power of skip-gram from vector of single words to vector of phrases but also further reduces the training time.

In the basic skip-gram model, the output layer computes the probability of a context word given a middle word by the softmax function:

\[
p(x_{t+j} | x_t) = \frac{\exp(v'_{x_{t+j}} ^\top v_{x_t})}{\sum_{w=1}^{W} \exp(v'_{w} ^\top v_{x_t})}
\]

where \( v_x \) is the input vector of word \( x \) and \( v'_x \) is the output vector of \( x \), and \( W \) is the total number of words in vocabulary. [30] suggest that it is impractical to compute the exact value of softmax in the output layer for the reason that the expense of computation is proportional to the size of vocabulary \( W \). As the training corpus gets bigger, the size of vocabulary increases, which lead to a long training time.

To address this problem, [30] proposed a simple yet powerful method called Negative sampling which is a simplified version of Noise Contrastive Estimation (NCE)[31]. Negative sampling redefines the following objective of training to replace the original \( \log P(x_{t+j}|x_t) \) term in the Skip-gram’s objective:

\[
\log \sigma(v'_{x_{t+j}} ^\top v_{x_t}) + \sum_{i=1}^{k} E_{x_i} \sim P_n(x) \log \sigma(-v'_{x_i} ^\top v_{x_t})
\]
where $\sigma(x) = \frac{1}{1+\exp(x)}$. The term on the right side represents the probability of $k$ negative samples $x_i$, which are chose from the original samples, being in the context of input word $x_t$. Therefore this objective function is use to distinguish the target word $w_{t+j}$ from the negative samples. And the such definition reduces the computational complexity by computing the distance between the target word and only a subset of the vocabulary. In the formula above, $P_n(x)$ is the probability distribution over the words in vocabulary, and [30] suggest that setting $P_n(x)$ as $3/4$rd power of the unigram distribution of $x$ gives the optimal result:

$$P_n(x) = \frac{1}{Z} \times \left( \frac{\text{count}(x)}{|\text{text}|} \right)^{3/4}$$

where $Z$ is a normalization constant.

In addition to Negative sampling, the extension of skip-gram proposed by [30] includes another method called Subsampling that improves the accuracy of the representations of rare words in the corpus and also produces a significant acceleration on the training time. They states that in a large corpus, there are many words that are frequently occurred but provides rather less information value comparing to the rare words, for instance, “in”, “the”, and “a” are the words that occur hundreds of millions of times in the corpus but the skip-gram benefit much less from such word because nearly every word will appear next to them. Therefore, the Subsampling method is proposed to discard each word $x_i$ in the training dataset with the probability defined as:

$$p(x_i) = 1 - \sqrt{\frac{t}{f(x_i)}}$$

where $f(x_i)$ is the frequency of the word $x_i$, and $t$ is a threshold usually set to be around $10^{-5}$. Such method balances the effect of rare and frequent words in the vocabulary.
Chapter 4

Financial Modeling

Financial modeling is the study of developing a mathematical model which abstracts the investment situations in real life. Such model is designed to represent and predict the performance of an investment derivatives, in particular, determining the price of an asset or evaluating the value of a company. There are different applications on financial modelling, for example, corporate finance applications is focusing on building a model that is base on the detailed information of a specific company or corporation, and such models are aiming on evaluating the performance of a certain company within a relatively long period; whereas quantitative finance typically aiming on development of models that deal with the overall and continuous behaviour of the market or a portfolio.

One of the most important application of quantitative finance is the modeling of the returns of different stocks which is also called “quantitative asset pricing”.

Obviously being able to predict the future price of different stocks will yield significant profits to investors, however, there has been a debate on whether such task is achievable because of the nature of the stock market. Efficient market hypothesis [16] and random walk [36] are two of the most famous hypothesis which made the argument that it
is impossible to predict a future price value of a specific stock. The efficient market hypothesis established an assumption that the movement of stock price is set efficiently in response to the available information as well as the rational expectation of the market. It suggests that the effect of all public information about the market or a specific stock have already been reflected on the price of the stocks. Moreover, the price of a stock is always fair and should only be affected by the news. Thus, it is impossible to trade stocks with knowing its future value in advance since the news itself is randomly published and is immediately reflected on the stock price once it is revealed. With a close link to efficient markets hypothesis, the random walk theory claims that the movement of stock market price could be represented as a statistical process consists of a set of random steps which is called a random walk, therefore the future price of a stock could not be predicted by analyzing the historical price data.

Even though evidence have been provided to prove such hypothesis, there are also a lot of facts exist that disagree with the statements made on those hypothesizes. For example, one of the most famous investors Warrent Buffett has consistently made investments that “beat the market”, which, according to the definition of efficient market hypothesis, is impossible. Moreover, researchers who do not believe in such hypothesis also pointed out that certain market event, such as the stock market crash when Dow Jones Industrial Average slumped over 20 percent in a single day, also shows that it is possible for an investor to purchase undervalued stocks to gain significant profit. In addition, other economists and investors have established experiments and research to disputes such hypothesis and show that the stock market is predictable to some degree. For instance, behavioral economists and psychologists argue that there are cognitive biases in financial markets, such as overconfidence, overreaction, and information bias, which leads to its imperfection. Those errors have been shown predictable and thus,
whoever reasons correctly should be able to create profit by trading on the stocks that have been undervalued because of other investors’ errors in reasoning.

Investors who believe the stock price is predictable have also been exploring technologies to actually show how to model the historical data and forecast the future price information. Traditionally there are two categories of methodologies: fundamental analysis and technical analysis. On top of those traditional methodologies, researchers also conducted experiments to take advantage of modern computers and make predictions using the artificial neural network.

4.1 Traditional stock market prediction

Fundamental analysis and technical analysis are two of the most widely used methodologies in the stock prediction industry. However, these two methods are analyzing the stock market in two completely opposite ways.

As the name suggests, fundamental analysis is the examination of fundamental factors of an investment asset. More specifically, the fundamental analysis considers all the potential effects that will impact the financial situations of a company or the industry or even the entire economy. For instance, the financial and management data of a company, as well as the balance of supply and demand within an industry. All of such financial aspects correlated to the business are concerned by fundamental analysts, from which the analysts gain insight of the company that enables them to make predictions on the company’s future performance.

In other words, fundamental analysis focuses on evaluating the economic well being of a financial entity instead of the actual price value. Such well being of a company is defined
as the *intrinsic value* of it. Intrinsic value is the "real value", or true value, a company worth, which according to the assumption of fundamental analysis, usually differs from the current price in the stock market. As a result, the gap between the true value and the market value of a company is where a fundamental analysis can create investment profit. This leads to the second major assumption of the fundamental analysis which states that the true value of a stock will be reflected by the market in a long term. So that if one estimates the intrinsic value of a company correctly and purchased the stock with a price below the estimation, it will eventually raise to the true price and generate returns.

In order to evaluate the intrinsic value of a company or a financial entity, fundamental analysts employ a top-down approach starts from the analysis of the overall economy down to the industry and finally to the specific company. The economy is the high-level guideline of the industries and companies within it; industries and companies usually benefit from the expansion of economy as much as they experiencing declines if the economy falls. One of the most important factor, which is kept tracked by the fundamental analyst, when evaluating the economy is the interest rate; for the reason that interest rate is an indicator of whether the central bank of a country tends to stimulate the investment or saving. After analysis of the overall economy, the nature of each industry is considered to discover the more detailed trend, since different industry group will behave differently within different economic environments. For instance, companies in a technology industry are likely to yield more earnings in an expanding economy, whereas the price of consumer businesses’ stock will have a more stable movement in a shrinking economy. Then, the analysis breaks down to a specific company and a list of criteria is reviewed, for example, a company’s business plan, the financial statement as well as management. Such criteria are used to identify the competitiveness of company
within its belonging industry sector. Lastly, these analyses over the top-down chain are summed up together to give a fundamental evaluation of a company.

In contrast to fundamental analysis, a technical analyst focuses on statistical data generated in the market and takes no consideration of any of the factors a fundamental analyst uses for evaluating a company. One essential principle guides the study of technical analysis is that the market price has already reflected all the information available to the public, which agrees to the similar premise claimed in the efficient market hypothesis. In addition, technical analysts believe that there is price trend \[20\] in the market, meaning that the financial market tends to move in a particular direction, i.e. moving up, down, or flat, within a certain period of time. Moreover, technical analysts have explored that the price movement in the market tends to repeat itself since the investors collectively tend to behave like their predecessors. Technical analysts believe there are patterns in historical market data and prediction of future stock price can be done base on such patterns. Therefore instead of studying a company’s intrinsic value, technical analysis collects the past information of stock, such as price and volume, and then search patterns among those statistical data in order to forecast future price.

There are many tools and techniques introduced by technical analyst to exploit certain patterns or forms from the historical data. One of the earliest methods was charting, a technique that plots the price or volume of stocks against time. Patterns that repeatedly occur can then be identified from the charts, for example header and shoulders, cup with handle or support and resistance. Such patterns are also called technical indicators, which are used by technical analysts to identify the whether a stock is trending, the possibility of the trend as well as the possible duration of the trend. On top of charting, methods with the mathematical transformation of market data are also applied, for example, moving average, or relative strength index.
Both fundamental analysis and technical analysis have their own advantages as well as disadvantages, fundamental focus on studying the intrinsic value a company in order to forecast a long-term return, whereas technical analysis applies various tools and methods to identify trends which purportedly allow investors to gain short-term profit. Although these two methodologies are completely contrast to each other, they are often combined together by professional investors to analyze the market.

4.2 Prediction with neural networks

With the development of modern computer technology, stock market analysts start to evaluate market with the help of the software systems. In particular, artificial neural network is one of the prominent technique with rapidly grown in popularity. ANN has been shown to be an *universal approximator* because of its powerful generalization ability, which is also the property that draws the attentions from stock market analysts. At the beginning, similar with technical analysis, pure market data were used as the input for the neural network, as technical analysts would like to benefit from the generalization ability to learn the pattern from the market data. Since with theory claiming the ability of universally matching any input-output pairs with appropriate network setting, the factors employed in fundamental analysis with a good mathematical representation could also be learned by a neural network.

Typically, there are two ways of applying neural networks to stock market prediction. One is to use the neural network as a curve fitting tool to predict the actual future value of time series data in the market, for instance, predicting the close price, lowest or highest trading price in a target day. The predicted market figures are then combined with certain trading strategy as a guideline to trigger the buy or sell action in the market.
Another method is using neural network as a classifier to predict discrete signal in the market, such method predicts either the most profitable action on a target date directly ("buy", "sell" or "hold"), or assigns a specific category to the target date and the output will have the form of "up" and "down". It has been shown in [21] that the classifier approach generates more reliable predictive results comparing to the linear regression approach since the neural network is better in classification problems.

Both of these two applications of neural networks involve complex design process since there is a good amount of parameters in the system need to be fine tuned in order to optimize the performance. For instance, the frequency of input data, i.e. either daily, weekly, or monthly stock data should be used to construct the training data; or time frame for the target date, how far should the prediction be made, 1-day, 2-days, or 5-days. All of which are choices that affect the accuracy of predictions. These parameters need to be tuned by experiments since they vary base on different nature of problems as well as the size of training data. [13] proposed an eight-step procedure of designing a neural network for forecasting financial time series data. The design procedure starts from the preparation of training samples, which involves selection of features, data collection, and preprocessing data. Selection of features is one of the most critical step in neural network designing since good feature could significantly improve the accuracy of prediction; data preprocessing includes feature transformation and scaling as good representations are easier to learn and each feature has their own range of domain, thus, every dimension of feature need to be scaled into a uniform domain, usually no raw data are directly fed into the neural network. The prepared samples are then divided into three different sets: training, testing and validation sets. The next stage of the designing procedure involves setup of the network structure as well as the training parameters. Network structure mainly refers to the number of hidden layers as well as the hidden units in each layer;
whereas training parameters include the number of epoch required for training, the size of mini-batch, learning rate in each epoch, weight decay or momentums, etc. The final stage is the implementation of the overall design and train the system.
Chapter 5

Predicting with Historical Price Data

One of the traditional and also widely used approach to analyze securities and make investment decisions is technical analysis, which focuses on the past price movement in the market in order to identify patterns that may suggest the direction of future price. Technical analysis relies on a set of technical indicators that are computed from the historical stock data. In the first part of this thesis, we construct our feature vectors using such technical indicators that are extracted from the Centre for Research in Security Prices (CRSP) database.

5.1 CRSP dataset

The historical stock price data used in this thesis are obtained from CRSP database which is designed for academical research and educational use and have proven to be highly accurate [22]. The CRSP stock files are developed by the Centre for Research
in Security Prices (CRSP), Booth School of Business, University of Chicago. CRSP database includes the daily, monthly and annually stock data as well as the market index data from the major US exchanges, including New York Stock Exchange (NYSE), NASDAQ and American Stock Exchange (AMEX). The original CRSP stock file was developed to contain the monthly stock prices and returns of NYSE dating from 1925, it then expand its coverage of NASDAQ and AMEX in 1962 and 1972.

Both individual security data and market index data are time series data, therefore, each entry in the data is tagged with a date. CRSP uses a calendar contains only the dates where a major US exchange is open for trading. The list of daily security data is synchronized with the CRSP calendar in the way that the \( n \)-th security record is associated with the \( n \)-th date in the CRSP calendar. CRSP stock files provide various type of data for individual securities and market index ranging from the stock identifier, numerical trading data, and other descriptors. Table 5.1 listed the descriptions of each field we selected from CRSP database as well as a stock record example of “GOOGLE INC” on 2006-01-03.

We download all available daily stock records from CRSP database during the period of 2006-01-03 to 2013-12-31, as same as the time frame of the financial news data we use in this thesis [27]. The query returns the records of 10430 stocks from American major stock markets. We then filter out all stocks that don’t have any related financial news base on the preprocess described in section 6.1, since those stocks have fewer attentions from the investors and are more likely to have missing data. After such filtering, we keep about 5 million records of 3170 stocks as our final dataset.

In order to access and search among the stock records easier and faster, we created a MySQL database to store all this records and identify each stock using the PERMNO
field since this is the permanent code that assigned to a stock discard the change of company name. For example, “APPLE COMPUTER INC” has changed its name to “APPLE INC” on 2007-01-11, and we use the PERMNO code of this company to clarify such name change event and collect all records associated to the same company. In addition, using such stock identifier help us to group the stock records and preprocess the data on the company base before constructing the feature vectors.

5.2 Preprocessing

Typically, raw dataset requires cleanup and preprocess before use. According to table 5.1, there are several special values defined for each field indicate missing data or special meaning of data. Therefore as the first stage of preprocessing, we handle all such special values from the original dataset. First we mark all missing or unavailable field in each record as NULL so that after we construct the feature vector samples, any sample with NULL value could be easily identified and removed. In the close prices field, since negative value means a substitute value of the unavailable closing price, we simply remove the negative sign and keep the absolute value in that field.

In the raw dataset, each field has their own domain range of values, and different company also have different value range in the same field. Therefore, before extracting features from the raw dataset, it is helpful to normalize the values in each field into a uniformed scale for a better generalization performance in a machine learning task. We first normalize each field in the company level by grouping the stock records using the PERMNO field described in the previous section. Then for each numerical field listed in table 5.1 excluding PERMNO, Date, and Company name, we preform the following
<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMNO</td>
<td>Stock identifier, CRSP’s unique and permanent issue identification number.</td>
<td>90319</td>
</tr>
<tr>
<td>Date</td>
<td>Date of the record with format “YYYY-MM-DD”.</td>
<td>2006-01-03</td>
</tr>
<tr>
<td>Company name</td>
<td>Full name of the company</td>
<td>GOOGLE INC</td>
</tr>
<tr>
<td>Close price</td>
<td>Closing price of the trading day. If the closing price is not available on any given trading day, the number in the price field has a negative sign to indicate that it is a bid/ask average and not an actual closing price. If neither closing price nor bid/ask average is available on a date, this field is set to zero.</td>
<td>435.23</td>
</tr>
<tr>
<td>Open price</td>
<td>Open price on that trading day.</td>
<td>422.52</td>
</tr>
<tr>
<td>Ask high</td>
<td>Highest trading price during the day, or the closing ask price on days when the closing price is not available, set to zero if this field is not available.</td>
<td>435.67</td>
</tr>
<tr>
<td>Bid low</td>
<td>Lowest trading price during the day, or the closing bid price on days when the closing price is not available. set to zero if this field is not available.</td>
<td>418.22</td>
</tr>
<tr>
<td>Number of trades</td>
<td>Total number of trades made during the day, set to 99 if not available.</td>
<td>70118</td>
</tr>
<tr>
<td>Holding period return</td>
<td>The change in total value of an investment in this stock over previous trading day, calculated as $\frac{p(t)}{p(t')} - 1$ where $p(t)$ is the closing price on day $t$ and $t'$ is the trading day before $t$.</td>
<td>0.04910</td>
</tr>
<tr>
<td>Market value weighted return</td>
<td>Return calculated base on market index of NASDAQ, AMEX and NYSE. Market index is the aggregated value produced by combining stocks in one stock market that intended to represent the entire stock market. Value-weighted market index is computed base on stock prices that are weighted by on their value in the market.</td>
<td>0.01641</td>
</tr>
<tr>
<td>Market equal weighted return</td>
<td>Similar with Market value weighted return except the market index is computed base on equal weighted stock value in the market</td>
<td>0.01098</td>
</tr>
<tr>
<td>S&amp;P index return</td>
<td>Return on the Standard &amp; Poor’s Composite index.</td>
<td>0.01643</td>
</tr>
</tbody>
</table>

Table 5.1: List of fields in each daily security record queried from CRSP database.
normalization, first calculate the mean value of the field:

\[
mean_s(f) = \frac{1}{n} \sum_d f_s(d)
\]

where \( f \) is the field name, the subscript \( s \) indicates that the corresponding field is associated with the stock with PERMNO \( s \), \( f_s(d) \) denotes the value of field \( f \) of company \( s \) on trading day \( d \), therefore \( \sum_d f_s(d) \) is the sum of all available values in field \( f \) associated with the stock \( s \), and \( n \) is the total number of records belong to the stock.

We then calculate the standard deviation of each field by:

\[
std_s(f) = \sqrt{E[(f_s(d) - mean_s(f))^2]}
\]

where \( E \) denotes the expected value. Finally, we normalize the field by calculating the standard score:

\[
z_s = \frac{f_s(d) - mean_s(f)}{std_s(f)}
\]

On thing should be mentioned is that before the training of neural network, we typically separate the original dataset into three different blocks, i.e. training set, validation set, and test set where training set represents the available data for training a predictive model like DNN, validation and test set represents the unseen data to the system. Therefore, the mean and variance should only be computed from the training set since the data in validation set and test set is not accessible in a real life system. In this thesis, since we are predicting time series data, the original dataset is divided base on the Date field in each record. For instance, all the records between 2006 and 2011 are used as training data, records from the year 2012 are used as validation data and records from the year 2013 are used as test data. Thus for each company, the mean and standard
deviation for each field is computed base on the records between 2006 and 2011, and the
data from 2012 to 2013 is normalized by the mean and standard deviation of data from
2006 to 2011.

The process described above normalizes the raw dataset on company bases. After stack
the feature vectors of all the company together, each dimension of the overall feature
vector will be normalized again using the same method.

5.3 Constructing feature vectors

After cleaning up and normalizing the raw dataset from CRSP, feature vectors could be
constructed from the preprocessed dataset and use for deep neural network training later
on. We extract three types of features from the dataset to form the samples of historical
stock data: (1) original fields provided by CRSP stock files; (2) technical indicators
derived from the original fields; (3) first and second order differences of all dimensions.

5.3.1 Original fields

Each of the fields listed in table 5.1 (with the exception of PERMNO, date, and company
name) contains useful information represent the behavior of individual stocks as well as
the market during a period of times. The first block of our feature vector is constructed
directly from the normalized value of the fields from the following list:

\[ F = \{ \text{Close price, Open price, Ask high, Bid low, Number of trades, Holding period return, Market value weighted return, Market equal weighted return, S&P index return} \} \]
Similar with normalizing the raw dataset, we construct the feature vectors on company base before stack each company’s feature vectors together to form the entire set of samples. First, each group of stock records with the same PERMNO code are sorted by their date. A list of $N$ continuous trading days’ records can be concatenated together on each field to form a feature vector. In this thesis, we concatenate 5 day’s record together. For instance, the part of closing price in the feature vector will have the form of:

$$F_{\text{closeprice}} = (P_{t-4}, P_{t-3}, P_{t-2}, P_{t-1}, P_t)$$

where $P_m$ denotes the closing price on date $m$. The overall feature vector of the original fields will have the form of:

$$F = (F_{\text{closeprice}}, F_{\text{openprice}}, \ldots, F_{Sk\text{Pindexreturn}})$$

where $F_{f\_name}$ is 5 continuous days’ value of field $f\_name$.

### 5.3.2 Technical indicators

The second block of the feature is inspired by the technical analysis discussed in chapter 4. We compute two types of technical indicators, which are widely used by the technical analyst, from our original dataset: exponential moving average (EMA) and relative strength index (RSI). These two features are also computed on the company base and normalized after combining all company’s samples.

**EMA** A moving average is used to smooth out the short-term bouncing and highlights the long-term moving trend of a time series data. Figure 5.1 shows an example of EMA of GOOGLE INC’s close price during a window of 100 trading days between 2006-01-03
and 2006-05-25. We compute EMA on closing price and open price for each stock. Using close price of a single stock as an example, we compute the moving average of close price on day $i$ as:

$$EMA_i = \alpha \times P_i + (1 - \alpha) \times EMA_{i-1}$$

where $P_i$ is the close price on day $i$. $\alpha$ is the decrease factor that exponentially decreases the impact of older close price have on the current $EMA_i$, we adopt the common method and set it as $\frac{2}{N+1}$ in which $N$ is the total number of available close price in the dataset. Since the above computation of EMA is a recursive computation, we set the base case of $EMA_1$, the $EMA$ of first day’s close price, as the average close price of the first 5 days in the time series. After computing the EMA for both close price and open price, we concatenate 5 day’s EMA together like we did for the original fields so that the feature vector of EMA will have the form:

$$EMA = (EMA_{t-4}, EMA_{t-3}, EMA_{t-2}, EMA_{t-1}, EMA_t)$$

**RSI** The relative strength index is a technical indicator developed by J. Welles Wilder [23], it is a momentum oscillator that measures the velocity and magnitude of the price movement. RSI is a value between 0 and 100 that is use as an indicator of price turning point, such that exceeding a certain *neutral* region indicates the change of price moving direction is upcoming. Figure 5.2 shows an example of RSI of Google’s close price. After the RSI value grows higher than the upper bound 70, it becomes a strong indicator of a price drop. Oppositely, if the RSI drops lower than 30, it indicates a price raise is likely to happen. We compute RSI of close price base on the following computations. First
calculate the upward change $U$ and downward change $D$:

$$U = \begin{cases} P_t - P_{t-1} & \text{if } P_t > P_{t-1} \\ 0 & \text{otherwise} \end{cases}$$

$$D = \begin{cases} 0 & \text{if } P_t \geq P_{t-1} \\ P_{t-1} - P_t & \text{otherwise} \end{cases}$$

where $P_t$ is the close price on day $t$. Then the RSI is computed as follow:

$$RSI = 100 - \frac{100}{1 + \frac{EMA(U)}{EMA(D)}}$$
Figure 5.2: Example of relative strength index of Google’s close price. The region between 30 and 70 is typically the neutral region. RSI higher than 70 indicates an overbought of the stock and price drop is about to happen, whereas RSI lower than 30 indicates an oversold of the stock and price rise is about to happen.

where \( EMA(U) \) and \( EMA(D) \) are the exponential moving averages of upward change and downward change respectively.

### 5.3.3 1st and 2nd order differences

In addition to the previous two feature vectors, we also compute first and second order differences of each field in \( \mathbf{F} \). Using close price as example, the first order difference is computed as follow:

\[
\Delta F_{\text{closeprice}} = (P_{t-4}, P_{t-3}, P_{t-2}, P_{t-1}) - (P_{t-5}, P_{t-4}, P_{t-3}, P_{t-2})
\]
And the second order difference is calculated by taking the difference between two adjacent values in each $\Delta F_{\text{closeprice}}$:

$$\Delta \Delta F_{\text{closeprice}} = (\Delta F_2, \Delta F_3, \Delta F_4) - (\Delta F_1, \Delta F_2, \Delta F_3)$$

We compute such first and second order differences for every fields in $F$ and append them to the previous two feature vectors together to get the final sample:

$$(F, \Delta F, \Delta \Delta F, \text{EMA, RSI})$$

5.4 Labeling

The final step of feature extraction is labeling the samples. Since the missing or unavailable data from original CRSP dataset are marked as NULL in the data preprocessing stage, we first remove all samples with NULL value in any dimension of the feature vectors. For the remaining samples, each of them is constructed by a set of features from trading day $t - 4$ to $t$, and we set the predicting target of each sample to be the price moving direction on the day after $t$. Therefore we use the holding period return value on day $t + 1$ to label each sample. Samples with day $t + 1$’s return value greater than zero will be label as “price up” and “price down” otherwise.
Chapter 6

Financial News Data

In addition to the technical analysis, which studies the pattern from pure market data, investors also use a wide variety of public available information to make their investment decisions. One of the most easy-to-access public available information is financial news. Financial news usually contains useful information of the market such as the financial report of a public company, analysis of the market or a set of specific stocks from professional investment organizations, or a news describing a remarkable event of the market or a company, etc. These financial articles not only indicate a company’s potential value but also reflects the investors’ confidence of the stock market. Therefore, the information from the financial articles would have certain impacts on the purchasing intention of investors as well as the market movements. The goal of the second part of this thesis is to extract useful features from the financial news and learn from these features in order to catch the trend of stock’s future moving direction.

The financial news data used in this thesis are provided by [27] which contains 106,521 articles from Reuters and 447,145 from Bloomberg. The news articles were published in the time period from October 2006 to December 2013.
6.1 Data preprocessing

Before extracting fixed-size feature vectors that are suitable to DNNs from financial news corpora, we need to preprocess the text data in order to find out all the related stocks for each financial news. Each financial news data entry has a title field, a body field, and a date of publication field as shown in table 6.1. First, for all financial articles, their body field is split into sentences. Next, each of these sentences needs to be labelled with its related stock. To do this, we first obtain a set of all company names from CRSP database [22] mentioned in the previous chapter. Since the company name of stocks from CRSP database are full company with all kinds of suffixes (such as “INC”, “CORP”, “LTD”, “PLC”, “LLC”, etc.), and such suffixes are usually omitted in the financial news, for instance, the company with name “Microsoft Corp” in CRSP stock file is usually referred as “Microsoft” in the financial news. Therefore, in order to get as much available samples as possible from the raw news content, we trim off all such kind of suffixes from the company names before we match them with the sentences. Nevertheless, we also select a list of companies that should keep their suffixes since such company name without suffix is a common word that will appear in a lot of irrelevant news, for example, companies with name “Express Inc” and “Unit Corp” without their suffixes will appear in many common sentences. We only keep those sentences that mention at least one public company. Each sentence is then labelled by the publication date of the original article and the mentioned stock name. It is possible that multiple stocks are mentioned in the same sentence. In this case, this sentence is labelled several times for each mentioned stock. We then group these sentences by the publication dates and the underlying stock names to form the samples. Each sample contains a list of sentences that were published on the same date and mentioned the same stock or
<table>
<thead>
<tr>
<th>Field name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2006-11-01</td>
</tr>
<tr>
<td>Title</td>
<td>Apple pushes Oricon to quit PC music downloads</td>
</tr>
<tr>
<td>Body</td>
<td>TOKYO (Reuters) - Oricon on Wednesday announced its exit from Japan’s PC music download market, becoming the first victim among local players to the surging popularity of Apple Computer Inc.’s iTunes music store . . .</td>
</tr>
</tbody>
</table>

Table 6.1: An example of financial news data entry.

company. Moreover, each sample is labelled as positive (“price-up”) or negative (“price-down”) based on its stock’s closing price on next trading day consulted from the CRSP financial database [22].

In the following sections, we will introduce our method to extract three types of features from the samples.

### 6.2 Bag of keywords feature

Bag-of-words (BoW) model is a common model in Nature Language Processing (NLP) that represents the text data, either a document or a sentence, as a bag of words appear in the data and use the frequency of each word as features; such features are used to train a classifier afterwards. In this thesis, the goal is to train a classifier to classify the input text into two categories: one indicates the probability of stock price drop, and the other one indicates the probability of stock price raise. Since the corpora of financial news used in this chapter contains more than 10 thousand words, which requires a feature vector with 10 thousand dimensions to represent the whole corpora if all of them are included in the vocabulary of the BoW model. Therefore, instead of using all words in the corpora as features, a Bag of keywords (BoK) are selected to form the feature vector.
The word embedding method in [1, 30] is used to select a list of words that have strong effect on the stock price. With the word embedding method, we use a small set of seed words that have obvious impact on the stock prices and then search through the vector representation of words to get a list of words that are “close” to these seed words.

We first compute the vector representations for all words occurring in the training set using the popular word2vec method \(^1\). Then, we manually select a small set of seed words based on two criteria: 1) the words frequently occurred in the title and content of financial articles; 2) the financial article with title or content containing such seed words describe a significant price change event of a stock. Following is a set of nine seed words we selected using these criteria:

\{\text{surge, rise, shrink, jump, drop, fall, plunge, gain, slump}\}

Next, we will repeat an iterative searching process to collect other useful keywords. In each iteration, we compute the cosine distance between other words occur in the training set and the seed words. The cosine distance represents the similarity between two words in the word vector model. For example, based on the pre-calculated word vectors, we have found other words, such as \text{rebound, decline, tumble, slowdown, climb, rally, weak, pullback, downtrend, boost}, which are very close to at least one of the seed words. And the top 10 most similar words are chosen and will be added back into the set of seed words at the end of each iteration. And the updated seed words will be used to repeat the searching process again to find another top 10 most similar words, the size of the seed words will keep increasing as we repeat the searching. In this way, we have searched all words occurring in the training set and kept the top 1,000 words (including the nine

\(^1\)https://code.google.com/p/word2vec/
seed words) that are close to each other as the keywords for our prediction task. The complete list of 1000 keywords are shown in Appendix A.

Since the way we repeat the searching process collects a set of words that are close to each other, no matter what set of seed words are used at the start point, as soon as they are chosen using the criteria described before, the resulting keywords set will be similar.

Finally, a list of 1000 keywords is constructed. And using the indexes of the list, each sample is represented by a 1000-dimension vector, where each dimension of the vector refers to the frequency of the corresponding keyword in the list (which is also the histogram representation). Since it is possible that some keywords appear more frequently than others, which will vanish the effect of other less frequent but useful keywords; it is common in text classification tasks to weigh terms by term frequency inverse document frequency or TF-IDF schemes instead of using term frequencies to avoid such case.

**TF-IDF** is a numerical statistic that is intended to reflect how important a word is to a sample in the training set. **Term frequency** is simply the number of times a term or word occurs in the sample. And **Inverse document frequency** is a factor that use to diminish the weight of a term occurs too frequently in the overall training set. There are several variants of TF and IDF weight. In the case of term frequency, which denotes as $TF(w,s)$, we use the simplest choice which is the number of times a word $w$ occurs in a sample $s$:

$$TF(w,s) = |\{w \in s\}|$$

For inverse document frequency, we use the form:

$$IDF(w,S) = \log \frac{N}{|\{s \in S : w \in s\}|}$$
where $S$ is the training set of samples, $N$ is the total number of samples in the training set, and $|\{s \in S : w \in s\}|$ represents the number of samples where the keywords $w$ appears. The $TF-IDF$ score is then calculated by multiplying these two terms:

$$TF-IDF(w, s, S) = TF(w, s) \ast IDF(w, S)$$

Base on this TF-IDF score, a 1000-dimension feature vector, called bag-of-keywords or BoK, is generated for each sample. Each dimension of the BoK vector is the $TF-IDF$ score computed for each selected keyword from the whole training corpus.

6.3 Polarity score feature

As well as including BoK feature vector to indicate the level of impact a financial news will have on the movement of stock price. We also want to extract feature that measures to which direction will the keyword affect the stock price move, i.e. either up or down, and by how much will the keyword affect it.

6.3.1 Pointwise mutual information

We further compute so-called polarity scores [33, 34] to measure how each keyword is related to stock movements. To compute such polarity score, we apply the concept of pointwise mutual information (PMI). PMI is a measure of relationship between two outcomes of random variables, for instance $X$ and $Y$, and it is defined using the formula:

$$PMI(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$
where \( P(x, y) \) is the joint probability of \( X=x \) and \( Y=y \), and \( P(x), P(y) \) is the probability of \( x \) and probability of \( y \) respectively. In the case of this thesis, PMI can be used to measure the relationship between a keyword and the category of positive or negative article. Therefore in the formula above, the outcome \( x \) could be replaced by the occurrence of a keyword \( w \) and \( y \) could be replaced by a category, i.e. either \( pos \) or \( nag \). And the PMI of a pair of keyword \( w \) and a positive category can be computed as:

\[
PMI(w, pos) = \log \frac{\frac{freq(w, pos)}{N}}{\frac{freq(w)}{N} \times \frac{freq(pos)}{N}}
\]

where \( freq(w, pos) \) denotes the frequency of the keyword \( w \) occurring in all positive samples, \( N \) denotes the total number of samples in the training set, and therefore \( \frac{freq(w, pos)}{N} \) denotes the probability of keyword \( w \) being in a positive article. Similarly, \( freq(w) \) denotes the total number of keyword \( w \) occurring in the whole training set, and \( \frac{freq(w)}{N} \) denotes the probability of keyword being \( w \); and \( freq(pos) \) denotes the total number of positive samples in the training set, and \( \frac{freq(pos)}{N} \) denotes the probability of an article being labeled as \( positive \). By cancelling out the \( N \)'s, the formula can be simplified to:

\[
PMI(w, pos) = \log \frac{freq(w, pos)}{freq(w) \times freq(pos)} \times N
\]

Then we can compute the PMI of a pair of keyword \( w \) and negative financial articles in the same way:

\[
PMI(w, nag) = \log \frac{freq(w, nag)}{freq(w) \times freq(nag)} \times N
\]

Furthermore, we calculate the polarity score for each keyword \( w \) as:

\[
PS(w) = PMI(w, pos) - PMI(w, neg).
\]
Obviously, the above polarity score $PS(w)$ measures how (either positively or negatively) each keyword is related to stock movements and by how much. Therefore, a keyword will have a positive polarity score if it has a strong relationship with the positive articles, oppositely, the polarity score will be negative if it has a strong relationship with the negative articles.

### 6.3.2 Assigning polarity score

Since it is possible that one sentence could include multiple stock names, a keyword’s polarity score in one sentence could have a different effect on the stocks mentioned in the same sentence, depending on the grammatical relation between the keyword and the stock name. Therefore, we need to decide how each keyword applies to a target stock in each sentence. To do this, we first use the Stanford parser [29] to generate a typed dependency parse that retrieves the grammatical relation between each two words in the sentence. Then, we use the grammatical relation to determine whether a stock name in the sentence is subject or not. If the target stock is not the subject of the keyword in the sentence, we assume the keyword is oppositely related to the underlying stock. As a result, we need to flip the sign of the polarity score. Otherwise, if the target stock is the subject of the keyword, we keep the keyword’s polarity score as it is. For example, the sentence:

> “Apple slipped behind Samsung and Microsoft in a 2013 customer experience survey from Forrester Research”

will be used in three samples, i.e. one labeled by Apple’s stock price, one labeled by Samsung’s stock price and the other one labeled by Microsoft’s stock price. For all these three samples, the word *slipped* will be identified as the keyword base on the
previous BoK feature. Then the whole sentence is passed to Stanford Parser to produce typed dependency. Based on the parsing result shown in figure 6.1, the grammatical relation between Apple and slipped is nsubj which means nominal subject; such relation indicates Apple is the subject of keyword “slipped”. While Samsung and Microsoft have a nmod relation (nominal modifiers) with “slipped” which indicate these two words are not subjects. Therefore, if this sentence is used as a sample for Apple, which is labeled by Apple’s stock price, the above polarity score of “slipped” is directly used. However, if this sentence is used as a sample for Samsung or Microsoft, the polarity score of “slipped” is flipped by multiplying $-1$.

Finally, the resultant polarity scores are multiplied to the TF-IDF scores to generate another 1000-dimension feature vector for each sample.
<table>
<thead>
<tr>
<th>Date</th>
<th>News Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-01-09</td>
<td>Investors dump RIM as Apple launches iPhone.</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>FACTBOX-Key facts on Apple’s new iPhone.</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>Apple introduces svelte multimedia iPhone.</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>Apple unveils iPhone.</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>Apple rolls out much-anticipated iPhone.</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>Apple shares up after Jobs introduces mobile phone.</td>
</tr>
</tbody>
</table>

Table 6.2: List of news title published on 2007-01-09.

6.4 Category tag feature

During the preprocess of the financial news data, we discovered that certain type of events are frequently described in the financial news, and the stock price will change significantly after the publication of such financial news. For example, table 6.2 shows a list of news published on 2007-01-09 related to Apple Inc. The titles show that this news is describing the same event which is the announcement of Apple’s new product iPhone. After the publication of this news, the stock price of Apple jumped from 92.57 per share on 2007-01-09 to 97.00 per share on the next day 2007-01-10. This is the type of new-product event we defined in this research.

In order to discover the impact of such specific event on the stock price, we further define a list of categories that may indicate a specific event or activity of a public company, which we call as category tags. In this thesis, we define the following ten categories:

- new-product
- acquisition
• price-rise
• price-drop
• law-suit
• fiscal-report
• investment
• bankrupt
• government
• analyst-highlights

We then use this list of category tags to create a new feature vector. Each category is first manually assigned with a few words that are closely related to the category. For example, we have chosen released, publish, presented, unveil as a list of seed words for the category new-product, which indicates the company’s announcement of new products. Similarly, we use the above word embedding model to automatically expand the above word list by searching for more words that have closer cosine distances with the selected seed words. In this work, we choose the top 100 words and assign them to each category, the complete list of category tags are shown in Appendix B.

After we have collected all keywords for each category, for each sample, we count the total number of occurrences of all words under each category, and then we take the logarithm to obtain a feature vector as

\[ V = (\log N_1, \log N_2, \log N_3, ..., \log N_c) \]
where $N_c$ denotes the total number of times the words in category $c$ appear in a sample. In the case where $N_c$ is zero, it is replaced by a large negative number, for example -99 in this work.
Chapter 7

Correlation Matrix

One limitation of the method discussed in the previous chapter is the amount of prediction could be made. Since the feature vectors we discussed in the previous chapter are mainly retrieved from financial news, the number of stock movement could be predicted in a day is restricted by the number of stocks mentioned in that day’s financial news. There are a large number of stocks trading in the market every day whereas we normally can only find a fraction of them mentioned in the daily financial news. Hence, for each date, the neural network model could only predict the stocks mentioned in the financial news on that date.

Our next goal is to make use of the prediction results produced by the DNN model and predict the stock prices that are not directly mentioned in any daily financial news. During the study of the historical stock data, we found that the prices of some stock are highly related to each other. It means there could be a certain strong relationship between two stocks, or even a group of stocks, that the prices of related stocks tend to move to the same direction. Figure 7.1 shows an example of such cases, the stock of two
companies "Cisco System Inc." and "Buckle Inc." are listed for trading during the same period of time, during which the price trend of these two stocks are very similar.

In this chapter, we propose a new method to predict more stocks that may not be directly mentioned in the financial news, which we call the unseen stocks. Here we propose to use a stock correlation graph, shown in Figure 7.2, to predict those unseen stocks. The stock correlation graph is an undirected graph, where each node represents a stock and the edge between two nodes represents the correlation between these two stocks. For example, if some stocks in the graph are mentioned in the news on a particular day, we first make the prediction to those stocks using the methods introduced in the previous chapters. Afterwards, the predictions are propagated along the edges in the graph to generate predictions for those unseen stocks.
Figure 7.2: Illustration of a part of correlation graph which contains 11 stocks. The symbol in the circle are the ticker name of the stock. The value along the edges are the correlation score of the two stocks connected by that edge.

7.1 Building the graph

We choose the top 5,000 stocks that satisfy the following two criteria from the CRSP database [22] to construct the correlation graph. First, the stock has, at least, one year’s trading records during the period between 2006 and 2012, this criteria is used to make sure that the stock is active enough in the stock market. Second, the stock has available closing price data in the year of 2013, which will be used as the test set in our experiment, this criteria is used to make sure that the stock will have labeled data to test the performance of our method.

At each time, any two stocks in the collection are selected to align their closing prices based on the related dates (between 2006/01/01 - 2012/12/31), and we only keep the stock pairs that have at least 252 days’ (number of trading days in one year) of overlap trading period. Then we first calculate the covariance, which is a measurement of how
much two random variables change together, of the two stocks. For two stocks $A$ and $B$, the covariance of them is calculated as:

$$
cov(A, B) = \frac{1}{n} \sum_{i=1}^{n} (A_i - E[A])(B_i - E[B])
$$

where $n$ is the total number of days $A$ and $B$ both have closing price available in the database, $A_i$ and $B_i$ are the day $i$’s closing price of stock $A$ and $B$ respectively; $E[A]$ denotes the expected value(or average value) of the close price of stock $A$. Then we calculate the correlation coefficient by:

$$
P(A, B) = \frac{\text{cov}(A, B)}{\sqrt{\text{cov}(A, A) \times \text{cov}(B, B)}}
$$

Where $\text{cov}(A, A)$ and $\text{cov}(B, B)$ are just covariance of the stock $A$ and $B$ themselves. The result value of $P(A, B)$ is between $-1$ and $1$, the greater this value is, the more likely the two stock will move to the same direction whereas the smaller the value is, the more likely the two stocks will move to the opposite directions. The computed correlation coefficient is attached to the edge connecting these two stocks in the graph, indicating their price correlation. The correlation coefficients are calculated for every pair of stocks with over one year’s overlapping on the trading period from the collection of 5,000 stocks. After calculating the correlation coefficient for all eligible stock pairs, we only keep the edges with an absolute correlation value greater than 0.8, all other edges are considered to be unreliable and pruned from the graph, a tiny fraction of the correlation graph is shown in Figure 7.2.

In this work, we represent such correlation graph by a 5000-by-5000 matrix, in which the index of row and column corresponding to a stock in the collection. Each element in this matrix is used to store the correlation coefficient value of two connected stock, for
example, the value in $i$-th row and $j$-th column of the matrix is set to be the correlation between the $i$-th and $j$-th stock in the collection. Obviously, this matrix will be a symmetric matrix since both the index of row and column in the matrix represents the same set of stocks.

### 7.2 Predict unseen stocks

In order to predict price movements of stocks that are not mentioned in the financial news, we take the prediction results of those stocks that are mentioned in a financial new from the DNN outputs and use the outputs as signals to propagation through the correlation graph.

We first construct a 5000-dimension vector $\mathbf{x}$. Each dimension corresponds to one stock in the correlation graph. The value of each dimension in this vector will be set to indicate the probability of price moving up or down. For the dimensions corresponding to stocks that have been mentioned in the financial news, we set the value by using the prediction output of DNN. Since the DNN outputs of each sample are the probabilities of two categories, i.e. stock price moving up or down, we first take the maximum value out of these two probabilities; and then distinguish the associated category by adding a negative sign on the value if it is the probability of price moving down. We set zeros for all other dimensions corresponding to unseen stocks. Then, the above graph propagation process can be mathematically represented as a matrix multiplication:

$$ \mathbf{x}' = \mathbf{A} \mathbf{x} $$
where $\mathbf{A}$ is the correlation matrix denoting all correlation weights in the graph. Of course, the graph propagation, i.e. matrix multiplication, may be repeated for several times:

$$x^{(n)} = \mathbf{A}^n x$$

where $n$ is the number of times the signal got propagated through the matrix. The prediction $x^{(n)}$ will eventually converge. After it converges, we put a threshold on each dimension and only keep the dimensions with highest absolute probability values. And the kept values in each dimension gave us another set of predictions of the stocks from the collection, where a positive value represents the probability of price moving up and a negative value represents the probability of price moving down.
Chapter 8

Experiments

The experiments are divided into two parts. The first part is designed to discover how different features will impact the performance of predictions on stock price movement, therefore, we train a deep neural network to learn the price and financial news features that are developed in Chapters 5 and 6, then the prediction results generated by learning different combinations of features are compared to explore the best combination of features. The second part is designed to show how we can make use of the correlation among stocks to further expand the results generated from the first part of the experiments; we take the prediction output from the first part and propagate them through the correlation matrix to make predictions on the stocks that are unseen from the original samples.

Before running the experiments, we need to divide the data we have into three separate sets: training set, validation set and test set. A training set is used to generate input samples for training the neural network. The validation set is a set of unseen data that is exclusive from the training set and is used to monitor the generalization performance of the DNN model and adjust learning rate during network training, validation set could
be used to eliminate the overfitting problem. A test set is another set of unseen data exclusive from both validation set and training set to test the final performance of the trained network. We divide both historical price data and financial news data into three sets based on their associated timestamp. We use data between 2006-10-01 and 2012-12-31 as the training set, data between 2013-01-01 and 2013-06-15 as the validation dataset and data between 2013-06-16 and 2013-12-31 are used as the test dataset.

8.1 Stock Prediction using DNNs

In the first part of experiments, we use DNNs to predict stock’s price movement based on a variety of features, namely producing a polar prediction of the price movement on next day (either price-up or price-down).

We first conduct a set of experiments with combinations of different network architectures and parameters in order to find the optimal setup for the network. In these first set of experiments, we use the price features introduced in Chapter 5 as our inputs. Moreover, the best result obtained in these experiments will be used as our baseline result in the next experiments. The performance of the neural network is measured by the error rate, which is calculated by counting the differences between the predicted category and the actual labelled category.

We mainly focus on tuning 4 types of parameters for the network: (a) input timeframe, which is the number of days’ historical data we add in the input samples; (b) DNN structure, which is the architecture of deep neural network that varies in the number of hidden layers as well as the number of hidden nodes in each layer; (c) learning rate, the constant factor we use when updating the weights of the network during training; (d) mini-batch size, which is the number of samples we feed into the network at once.
Tables 8.1, 8.2, 8.3 and 8.4 list some of the experimental results based on each of the parameters. In these experiments, we used the maximum of 50 epochs to train the network. Validation dataset is used to monitor the performance of the trained network in each epoch and adjusts the learning rate accordingly in order to reduce overfitting. In each epoch, using cross entropy as a validation criteria, after the network’s weights has been updated by back propagation algorithm, we test the performance of the updated network by feed the samples in validation dataset to the network; if the result cross entropy is too much higher (10%) that the one produced by the network trained in previous epoch, we discard the current epoch by reverting the updated weight and reduce the learning rate and retrain the network. The training process is finished if the learning rate has been reduced too small or all 50 epochs have been finished. Base on the experimental results, we select the network with 5 days’ input timeframe, 3 layers of 1024 hidden units structure, learning rate 0.005 and 200 mini-batch size as the optimal network, which will be used as our network system for the following experiments.

<table>
<thead>
<tr>
<th>input timeframe</th>
<th>DNN structure</th>
<th>learning rate</th>
<th>mini-batch size</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>50.09%</td>
</tr>
<tr>
<td>14 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>48.96%</td>
</tr>
<tr>
<td>7 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>48.23%</td>
</tr>
<tr>
<td><strong>5 days</strong></td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td><strong>47.64%</strong></td>
</tr>
<tr>
<td>4 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>48.09%</td>
</tr>
<tr>
<td>3 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>48.45%</td>
</tr>
<tr>
<td>2 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>50.25%</td>
</tr>
</tbody>
</table>

**Table 8.1:** Experiments on different input timeframes. We show that given other parameter unchanged, the input samples containing 5 days’ data generates the best performance.

After we have selected the optimal network setups, we run another set of the experiments to test the different combinations of feature vectors to find the features that have the
<table>
<thead>
<tr>
<th>input timeframe</th>
<th>DNN structure</th>
<th>learning rate</th>
<th>mini-batch size</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 days</td>
<td>[500]</td>
<td>0.005</td>
<td>200</td>
<td>50.00%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024]</td>
<td>0.005</td>
<td>200</td>
<td>50.01%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>48.79%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>47.64%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>47.64%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024 1024 1024]</td>
<td>0.005</td>
<td>200</td>
<td>47.62%</td>
</tr>
</tbody>
</table>

Table 8.2: Experiments on different DNN structures. Number of elements in the square brackets represent the number of hidden layers used in the neural networks, the value of each element denotes the number of hidden units in the corresponding hidden layer. Even though the networks with 3, 4 or 5 layers of 1024 hidden units generate almost the same performance, we choose the structure of 3 layers as the optimal one as it has the least computational cost.

<table>
<thead>
<tr>
<th>input timeframe</th>
<th>DNN structure</th>
<th>learning rate</th>
<th>mini-batch size</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.01</td>
<td>200</td>
<td>49.89%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td><strong>0.005</strong></td>
<td>200</td>
<td><strong>47.64%</strong></td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.001</td>
<td>200</td>
<td>47.64%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.0008</td>
<td>200</td>
<td>48.00%</td>
</tr>
</tbody>
</table>

Table 8.3: Experiments on different learning rates. We show that with validation, network with learning rate set to 0.005 produces the best performance and converge fast.

<table>
<thead>
<tr>
<th>input timeframe</th>
<th>DNN structure</th>
<th>learning rate</th>
<th>mini-batch size</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>100</td>
<td>48.89%</td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td><strong>200</strong></td>
<td><strong>47.64%</strong></td>
</tr>
<tr>
<td>5 days</td>
<td>[1024 1024 1024]</td>
<td>0.005</td>
<td>1000</td>
<td>47.92%</td>
</tr>
</tbody>
</table>

Table 8.4: Experiments on different mini-batch size.

most significant contributions to predictive accuracy. Base on the results from the previous sets of experiments, we use the historical price features alone to create the baseline. Various features derived from the financial news described in Chapter 6 are added on top of it. As shown in Table 8.5, the features derived from financial news can significantly improve the prediction accuracy and we have obtained the best performance
(an error rate of 43.13%) by using all the features discussed in Chapters 5 and 6.

In addition, we also compare our proposed feature with the structured events feature proposed in [27]. We extracted the structured event features from our data and Table 8.5 shows that our proposed features produce better performance in predicting a pool of individual stock prices.

<table>
<thead>
<tr>
<th>feature combination</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>random guess</td>
<td>50.23%</td>
</tr>
<tr>
<td>price</td>
<td>47.64%</td>
</tr>
<tr>
<td>price + BoK</td>
<td>46.02%</td>
</tr>
<tr>
<td>price + BoK + PS</td>
<td>43.96%</td>
</tr>
<tr>
<td>price + BOK + CT</td>
<td>45.86%</td>
</tr>
<tr>
<td>price + PS</td>
<td>45.00%</td>
</tr>
<tr>
<td>price + CT</td>
<td>46.10%</td>
</tr>
<tr>
<td>price + PS + CT</td>
<td>46.03%</td>
</tr>
<tr>
<td>price + BoK + PS + CT</td>
<td>43.13%</td>
</tr>
<tr>
<td>structured events</td>
<td>44.79%</td>
</tr>
</tbody>
</table>

Table 8.5: Stock prediction error rates on the test set.

In order to test the significance of our results, we applied the McNemar’s Test [44], which is a statistical test used on paired nominal data. McNemar’s test is applied on a 2-by-2 contingency table (shown in table 8.6), which tabulates the outcomes of two tests on a sample of n subjects. In McNemar’s test, it defines a null hypothesis that the marginal probabilities in the contingency table are the same, i.e. \( p_a + p_b = p_a + p_c \) and \( p_c + p_d = p_b + p_d \). Whereas an alternative hypothesis is defined such that the marginal probabilities are not the same. In addition, McNemar test statistic is defined as:

\[
\chi^2 = \frac{(b - c)^2}{b + c}
\]
Figure 8.1: Probability density function of Chi-squared distribution. \( k \) denotes the degree of freedom.

with the assumption that the null hypothesis is true. \( \chi^2 \) has a *chi-squared distribution* (shown in figure 8.1) with 1 degree of freedom. If \( \chi^2 \) is *extreme* in the distribution, the test will reject the null hypothesis and show that the marginal probabilities are *significantly* different from each other in favour of the alternative hypothesis. The level of extremeness of \( \chi^2 \) is measured by a *p-value* with respect to a significance level \( \alpha \) (traditionally set to 0.05 or 0.001). In our case, the *p-value* is the probability of the variable greater than the computed \( \chi^2 \) in the chi-square distribution, i.e. \( P(x > \chi^2) \).

<table>
<thead>
<tr>
<th></th>
<th>Test 2 positive</th>
<th>Test 2 negative</th>
<th>Row total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1 positive</td>
<td>a</td>
<td>b</td>
<td>a + b</td>
</tr>
<tr>
<td>Test 1 negative</td>
<td>c</td>
<td>d</td>
<td>c + d</td>
</tr>
<tr>
<td>Column total</td>
<td>a + c</td>
<td>b + d</td>
<td>n</td>
</tr>
</tbody>
</table>

Table 8.6: 2 x 2 contingency table applied in McNemar’s test.

In our experiments, we test the significance of different models by comparing the result of different feature combinations with the result of random classifier (the results are generated by random guess). In particular, we define the null hypothesis as follow: *the predictive performances of the DNN models with different input feature combinations*
would be the same as the random classifier. For each feature combination’s result listed in table 8.5, we created the 2x2 contingency table with the results generated by the random classifier. For example table 8.7 shows the 2x2 contingency table of the results from random classifier and the DNN model with only price data. Each cell in the table shows the matched results count of these two classifier; for instance, the top left cell shows the number of results predicted correctly by both of the random classifier and DNN model with price input feature. The McNemar test statistic $\chi^2$ value computed using this table is 10.90 which has a p-value 0.00096. After constructing such 2x2 contingency table for each DNN model with different feature combinations, we compute their $\chi^2$ and p-value, results are shown in table 8.8. The p-values shown in table 8.8 are significantly lower than the typical $\alpha$ value of 0.001, which provides a strong evidence to reject the null hypothesis of random guess.

<table>
<thead>
<tr>
<th></th>
<th>DNN w/ price correct</th>
<th>DNN w/ price wrong</th>
<th>Row total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random guess correct</td>
<td>2,860</td>
<td>2,609</td>
<td>5,469</td>
</tr>
<tr>
<td>Random guess wrong</td>
<td>2,853</td>
<td>2,589</td>
<td>5,480</td>
</tr>
<tr>
<td>Column total</td>
<td>5,713</td>
<td>5,198</td>
<td>10,911</td>
</tr>
</tbody>
</table>

Table 8.7: 2 x 2 contingency table of the results from random classifier and the DNN model with price data.

8.2 Predict Unseen Stocks via Correlation

In the first part of the experiments, since the number of samples are restricted by the available financial news related to the stock. On each trading day, only the stocks with financial news published on the previous day will be predicted. Therefore in the second part of the experiments, we take the predictions produced by the neural networks and
<table>
<thead>
<tr>
<th>feature combination</th>
<th>error rate</th>
<th>$\chi^2$</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>random guess</td>
<td>50.23%</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>price</td>
<td>47.64%</td>
<td>10.90</td>
<td>0.00096</td>
</tr>
<tr>
<td>price + BoK</td>
<td>46.02%</td>
<td>36.09</td>
<td>$1.88 \times 10^{-9}$</td>
</tr>
<tr>
<td>price + BoK + PS</td>
<td>43.96%</td>
<td>78.81</td>
<td>$6.83 \times 10^{-19}$</td>
</tr>
<tr>
<td>price + BOK + CT</td>
<td>45.86%</td>
<td>36.09</td>
<td>$1.88 \times 10^{-9}$</td>
</tr>
<tr>
<td>price + PS</td>
<td>45.00%</td>
<td>55.19</td>
<td>$1.09 \times 10^{-13}$</td>
</tr>
<tr>
<td>price + CT</td>
<td>46.10%</td>
<td>38.94</td>
<td>$4.37 \times 10^{-10}$</td>
</tr>
<tr>
<td>price + PS + CT</td>
<td>46.03%</td>
<td>36.09</td>
<td>$1.88 \times 10^{-9}$</td>
</tr>
<tr>
<td>price + BoK + PS + CT</td>
<td>43.13%</td>
<td>99.52</td>
<td>$1.94 \times 10^{-23}$</td>
</tr>
<tr>
<td>structured events [27]</td>
<td>44.79%</td>
<td>60.70</td>
<td>$6.64 \times 10^{-15}$</td>
</tr>
</tbody>
</table>

Table 8.8: Results of the McNemar’s Test of our models.

make another set of predictions on the stocks that do not have any related financial news using the correlation matrix.

Here we group all outputs from DNNs based on the dates of all samples on the test set. For each date, we create a vector $\mathbf{x}$ based on the DNN prediction results for all observed stocks and zeros for all unseen stocks, as described in Chapter 7. Then, the vector is propagated through the correlation graph to generate another set of stock movement prediction. We may apply a threshold on the propagated vector to prune all low-confidence predictions. The remaining ones may be used to predict some stocks unseen on the test set. The prediction of all unseen stocks is compared with the actual stock movement on next day.

Experimental results are shown in Figure 8.2, where the left y-axis denotes the prediction accuracy and the right y-axis denotes the percentage of stocks predicated out of all 5000 per day under each pruning threshold. For example, using a large threshold (0.9), we may predict with an accuracy of 52.44% on 354 extra unseen stocks per day, in addition
Figure 8.2: Predict unseen stocks via correlation to predicting only 110 stocks per day on the test set.
Chapter 9

Conclusions

In this thesis, we proposed a model to predict the moving direction of future stock prices using the deep learning techniques. With the deep neural network as our predictive model, we mainly leverage two types of input features: one is the price features computed from historical market data; the other one is natural language features extracted from online financial news based on the popular word embedding method. Our experiments tested different combinations of input features and have shown that the financial news is very useful in stock prediction and the features from news can significantly improve the prediction accuracy on a standard financial data set. Moreover, we proposed a correlation matrix which makes use of the underlying relationships among stocks to expand our predictive results.

However, being able to predict the moving direction of stock prices does not equal to beating the market. On one hand, a certain type of trading strategy is required to combine with the predictive results in order to make the real profit out of it; on the other hand, the amount of price change is also another important factor that is used to compute the return of investments. Therefore, the development of a set of trading
strategy or a system that predicts the value of price change could be the future direction of this research.
Bibliography


[26] Jianfeng Si, Arjun Mukherjee, Bing Liu, Sinno Jialin Pan, Qing Li, and Huayi Li. Exploiting social relations and sentiment for stock prediction. In *Proceedings of the


[38] Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Lirong Dai. The fixed-size ordinally-forgetting encoding method for neural network language models. In


Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2016), June 2016.
## Appendix A

### Bag of Keywords Features

<table>
<thead>
<tr>
<th>abandon</th>
<th>abates</th>
<th>abating</th>
<th>absence</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerated</td>
<td>accelerates</td>
<td>accelerateshtml</td>
<td>accelerating</td>
</tr>
<tr>
<td>accelerations</td>
<td>accumulated</td>
<td>achieve</td>
<td>add</td>
</tr>
<tr>
<td>advance</td>
<td>advanced</td>
<td>advances</td>
<td>advanceshtml</td>
</tr>
<tr>
<td>affect</td>
<td>affection</td>
<td>aftermath</td>
<td>albeit</td>
</tr>
<tr>
<td>allaying</td>
<td>amid</td>
<td>amongst</td>
<td>anaemic</td>
</tr>
<tr>
<td>anemic</td>
<td>anomalies</td>
<td>appreciable</td>
<td>appreciate</td>
</tr>
<tr>
<td>appreciates</td>
<td>appreciation</td>
<td>approximate</td>
<td>automatic</td>
</tr>
<tr>
<td>awhile</td>
<td>back</td>
<td>backdrop</td>
<td>backstop</td>
</tr>
<tr>
<td>balloon</td>
<td>ballooned</td>
<td>ballooning</td>
<td>bask</td>
</tr>
<tr>
<td>batters</td>
<td>bear</td>
<td>bearishness</td>
<td>beat</td>
</tr>
<tr>
<td>beats</td>
<td>blanket</td>
<td>bleak</td>
<td>blip</td>
</tr>
<tr>
<td>bolster</td>
<td>bolstering</td>
<td>boost</td>
<td>boosting</td>
</tr>
<tr>
<td>breakout</td>
<td>bring</td>
<td>budget</td>
<td>bull</td>
</tr>
<tr>
<td>buoy</td>
<td>buoyant</td>
<td>buoying</td>
<td>buoys</td>
</tr>
<tr>
<td>calibrating</td>
<td>calibration</td>
<td>calming</td>
<td>calms</td>
</tr>
<tr>
<td>catalyst</td>
<td>cause</td>
<td>caution</td>
<td>cautiousness</td>
</tr>
<tr>
<td>centerpiece</td>
<td>centimes</td>
<td>challenges</td>
<td>changes</td>
</tr>
<tr>
<td>cheer</td>
<td>chokes</td>
<td>choppy</td>
<td>climb</td>
</tr>
<tr>
<td>climbing</td>
<td>come</td>
<td>clinch</td>
<td>cloth</td>
</tr>
<tr>
<td>cncpiyoy</td>
<td>conflagration</td>
<td>conflicting</td>
<td>complacency</td>
</tr>
<tr>
<td>conditions</td>
<td>constrain</td>
<td>constraining</td>
<td>confounding</td>
</tr>
<tr>
<td>considerably</td>
<td>contracted</td>
<td>contracting</td>
<td>constriction</td>
</tr>
<tr>
<td>continuation</td>
<td>contribute</td>
<td>converging</td>
<td>contraction</td>
</tr>
<tr>
<td>contrasting</td>
<td>cooled</td>
<td>cooling</td>
<td>cooling</td>
</tr>
<tr>
<td>cooldown</td>
<td>cratering</td>
<td>cramps</td>
<td>cramps</td>
</tr>
<tr>
<td>correction</td>
<td>countering</td>
<td>crimping</td>
<td>crimping</td>
</tr>
<tr>
<td>crimped</td>
<td></td>
<td></td>
<td>crises</td>
</tr>
</tbody>
</table>

96
rallying
ravages
rebalancing
recalibrate
recoveries
reductions
relatively
resiliency
restrained
resurgent
retreat
revalue
revive
rise
robustness
rout
sagging
scorch
seloff
sequestration
shave
shortfalls
shuddered
sinks
skyrocketing
slides
slippage
slow
slower
sluggish
slumps
snowballed
soaring
solace
spasm
spikes
spiralling
spur
sputters
stabilizing
stagnates
stalls
stoking
streaks
strengthens
stumble
rangebound
reaccelerate
rebasings
recede
recovery
reminiscent
resilient
restraints
retarding
retreated
reverberate
revives
rises
rocketing
routs
sank
see
seloffs
severe
shock
shrink
shutout
skid
slash
sliding
slipped
slowdown
slowing
slump
snap
snuffed
soars
solid
spate
spiking
spite
spurred
stabilization
stable
stagnating
stern
strains
strength
strengthenshtml
stunning
rapid
reacceleration
rebound
recession
reduce
reinforce
rename
respectable
restrict
retesting
reverse
revitalize
rippled
risks
rose
runaway
sapping
seeped
sequester
shake
shortage
shrinking
significant
skyrocket
slid
sightly
slips
slowed
slowest
slumped
snapped
snuffing
soar
softening
somewhat
spending
spoil
spiral
spitting
spurt
stabilized
stagnate
stalling
stimulating
streak
strengthened
strong
stuttering
<table>
<thead>
<tr>
<th>stutters</th>
<th>subdued</th>
<th>subpar</th>
<th>subsector</th>
</tr>
</thead>
<tbody>
<tr>
<td>subside</td>
<td>subsidies</td>
<td>subsidy</td>
<td>substantial</td>
</tr>
<tr>
<td>supercommittee</td>
<td>supplant</td>
<td>supplementary</td>
<td>surge</td>
</tr>
<tr>
<td>surges</td>
<td>surging</td>
<td>surpass</td>
<td>surpassed</td>
</tr>
<tr>
<td>surplus</td>
<td>surpluses</td>
<td>surprise</td>
<td>sustaining</td>
</tr>
<tr>
<td>sustained</td>
<td>swell</td>
<td>swelling</td>
<td>swells</td>
</tr>
<tr>
<td>swoon</td>
<td>swooned</td>
<td>swooning</td>
<td>sympathy</td>
</tr>
<tr>
<td>tailwind</td>
<td>take</td>
<td>tanked</td>
<td>target</td>
</tr>
<tr>
<td>tariffs</td>
<td>temper</td>
<td>tempered</td>
<td>tempering</td>
</tr>
<tr>
<td>temptation</td>
<td>tenious</td>
<td>tepid</td>
<td>thawing</td>
</tr>
<tr>
<td>threatening</td>
<td>thresholds</td>
<td>throes</td>
<td>throw</td>
</tr>
<tr>
<td>timidity</td>
<td>timnochng</td>
<td>topped</td>
<td>topping</td>
</tr>
<tr>
<td>touching</td>
<td>trail</td>
<td>trailing</td>
<td>trajectories</td>
</tr>
<tr>
<td>transformation</td>
<td>translate</td>
<td>trend</td>
<td>trendline</td>
</tr>
<tr>
<td>trickled</td>
<td>trims</td>
<td>tucpiy</td>
<td>tumble</td>
</tr>
<tr>
<td>tumblehtml</td>
<td>tumbles</td>
<td>tumblehtml</td>
<td>tumbling</td>
</tr>
<tr>
<td>turbulence</td>
<td>turmoil</td>
<td>turn</td>
<td>turnaround</td>
</tr>
<tr>
<td>unanticipated</td>
<td>uncertain</td>
<td>uncertainties</td>
<td>underpricing</td>
</tr>
<tr>
<td>undermine</td>
<td>undermining</td>
<td>underpin</td>
<td>unevenness</td>
</tr>
<tr>
<td>undershooting</td>
<td>undertones</td>
<td>unspectacular</td>
<td>unstable</td>
</tr>
<tr>
<td>unsettles</td>
<td>unsettling</td>
<td>upswing</td>
<td>uptick</td>
</tr>
<tr>
<td>upheaval</td>
<td>upheavals</td>
<td>upward</td>
<td>usbodefn</td>
</tr>
<tr>
<td>uptrend</td>
<td>upturn</td>
<td>varied</td>
<td>various</td>
</tr>
<tr>
<td>valuationshtml</td>
<td>vanish</td>
<td>vitality</td>
<td>volatilities</td>
</tr>
<tr>
<td>versus</td>
<td>victoryhtml</td>
<td>vulnerable</td>
<td>waiver</td>
</tr>
<tr>
<td>vulnerabilities</td>
<td>vulnerability</td>
<td>wanes</td>
<td>waneshtml</td>
</tr>
<tr>
<td>wane</td>
<td>waned</td>
<td>wasteful</td>
<td>wave</td>
</tr>
<tr>
<td>warning</td>
<td>wastage</td>
<td>weakening</td>
<td>weakening</td>
</tr>
<tr>
<td>weaken</td>
<td>weakened</td>
<td>weakness</td>
<td>weathering</td>
</tr>
<tr>
<td>weakenshtml</td>
<td>weaker</td>
<td>wholesaling</td>
<td>widen</td>
</tr>
<tr>
<td>whipsaw</td>
<td>whipsawing</td>
<td>winning</td>
<td>winsite</td>
</tr>
<tr>
<td>widens</td>
<td>winhtml</td>
<td>wobble</td>
<td>wobbles</td>
</tr>
<tr>
<td>withstanding</td>
<td>withstands</td>
<td>worsened</td>
<td>worsening</td>
</tr>
<tr>
<td>worries</td>
<td>worsen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Category Tag Features

B.1 Category: new-product

<table>
<thead>
<tr>
<th>abandon</th>
<th>abolish</th>
<th>add</th>
<th>adopt</th>
<th>adopted</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>amend</td>
<td>announce</td>
<td>announced</td>
<td>announces</td>
</tr>
<tr>
<td>approve</td>
<td>assessment</td>
<td>begin</td>
<td>clarify</td>
<td>commence</td>
</tr>
<tr>
<td>complete</td>
<td>conclude</td>
<td>consider</td>
<td>continue</td>
<td>decide</td>
</tr>
<tr>
<td>defer</td>
<td>delay</td>
<td>detailed</td>
<td>devise</td>
<td>discuss</td>
</tr>
<tr>
<td>dominancehtml</td>
<td>enable</td>
<td>enact</td>
<td>expedite</td>
<td>facilitate</td>
</tr>
<tr>
<td>file</td>
<td>finalize</td>
<td>formulate</td>
<td>implement</td>
<td>implementing</td>
</tr>
<tr>
<td>incorporate</td>
<td>initiate</td>
<td>introduce</td>
<td>introduced</td>
<td>introduces</td>
</tr>
<tr>
<td>introducing</td>
<td>invite</td>
<td>macroprudential</td>
<td>oblige</td>
<td>omts</td>
</tr>
<tr>
<td>outline</td>
<td>outlined</td>
<td>outlines</td>
<td>outlining</td>
<td>overhaul</td>
</tr>
<tr>
<td>overhauled</td>
<td>overhauling</td>
<td>overhauls</td>
<td>plan</td>
<td>postpone</td>
</tr>
<tr>
<td>propose</td>
<td>publication</td>
<td>publish</td>
<td>published</td>
<td>pursue</td>
</tr>
<tr>
<td>readiness</td>
<td>reconsider</td>
<td>rejuvenate</td>
<td>release</td>
<td>released</td>
</tr>
<tr>
<td>releases</td>
<td>releasing</td>
<td>reopen</td>
<td>report</td>
<td>reshape</td>
</tr>
<tr>
<td>resubmit</td>
<td>resume</td>
<td>revamp</td>
<td>revamping</td>
<td>review</td>
</tr>
<tr>
<td>revise</td>
<td>revisit</td>
<td>revive</td>
<td>rt</td>
<td>schedule</td>
</tr>
<tr>
<td>scrapped</td>
<td>seek</td>
<td>shelf</td>
<td>shelved</td>
<td>simplify</td>
</tr>
<tr>
<td>start</td>
<td>submit</td>
<td>submitted</td>
<td>suspend</td>
<td>take</td>
</tr>
<tr>
<td>unqualified</td>
<td>undertake</td>
<td>unveil</td>
<td>unveiled</td>
<td>unveiling</td>
</tr>
<tr>
<td>unveils</td>
<td>update</td>
<td>updated</td>
<td>updates</td>
<td>updating</td>
</tr>
</tbody>
</table>

B.2 Category: acquisition
| accumulate | acquire | acquired | acquires | acquiring |
| construction | add | advertise | align | amass |
| approve | atacadao | bid | broaden | buy |
| buying | collaborate | collect | combine | compete |
| complement | consolidate | contribute | convertible | cooperate |
| create | develop | dispose | disposed | disposes |
| disposing | distribute | divest | divested | divesting |
| divestment | divests | donate | enable | exchangeable |
| exploring | freedompop | inspect | integrate | integrating |
| ious | liquidate | ltn | merge | merges |
| merging | modernize | noncore | offer | offers |
| offload | offloading | option | outperform | owning |
| proceed | promissory | purchase | purchases | rebuild |
| recapitalize | receive | redeem | refinance | refinanced |
| refund | reinvest | renegotiate | reorganize | reorganizing |
| repaid | repay | repaying | repayments | reposition |
| reschedule | resell | restructure | restructured | sale |
| sell | shedding | simplify | spinoff | sterilizes |
| stick | streamline | streamlining | subordinated | takeover |
| transfer | underperform | underwrite | unload | unloading |

### B.3 Category: price-rise

| advanced | advances | advancing | beating | bleak |
| blew | blowing | blows | bolstering | boosting |
| brighter | buoyant | caught | climb | climbed |
| climbing | climbs | cloudy | comeback | continued |
| curbing | decent | deteriorated | deteriorates | emergence |
| expansion | flurry | gained | gains | gloomy |
| growth | healthy | high | higher | highest |
| highs | improved | improves | improving | increasing |
| intensification | jump | jumped | jumping | jumps |
| lackluster | midst | missing | moderating | peak |
| peaked | peaks | record | rallying | recovery |
| rebound | rebounding | record | recovering | recovery |
| resilient | resurgence | retreated | retreating | revival |
| rise | rises | rising | robust | rocketing |
| rose | rosy | skyrocketing | slew | soar |
| soaring | softer | solid | spate | spiking |
| spiraled | spiraling | spiralling | stabilizes | stagnates |
| strengthening | strong | stronger | strongest | surge |
| surged | surges | surging | surpassing | tearing |
| threatening | topping | trailing | trough | turnaround |
B.4 Category: price-drop

correction deaccelerated decelerating deceleration decline
decreases deepening depress depressed deteriorating
digit digits diminished diminishes diminishing
dip drop dropped dropping drops
eroded erodes eroding fade fall
falling falls fell fluctuate hover
hovered hulled hurting lackluster less
lessened lessens level low lower
lowering lowest lows moderated moderating
more outweighs plummet plunge plunge
plunged plunges plunging plunging pullback
quickened rather recession record recovery
reduces relapse relapsing retreat retreated
selloff shock sink slid slide
slide sliding slip slipped slowdown
slowly slowing sluggish slump slumped
slumpings slumps softer spike stabilizing
stagnating slumps softer spike stabilizing
weakening weakened weakest weakness whammy
worse

B.5 Category: law-suit

accusing activevideo affidavit allegations allege
alleged alleges alleging applewhite arbitrator
arraigned arthrex asserted bailiff beiswenger
blower bratz case cases claim
claimed contended copyright cordace counterclaim
counterclaims countersued countsuit criminal declaratory
defamation defrauded dismissed document
documents earthgrains ecuadoreans explanatory fbi
filed filing fuhu guilty hoefner
idna indictment indictments infringement infringements
infringers infringing innovatio invalidation ivi
jury kolon lawsuit lawsuits
leadscope libel listing lodged macrosolve
mattel melendres meritless mga ntp
nuvasive obscenity patent petition plaintiffs
presentation proceedings prosecutors prospectus racketeering
### B.6 Category: fiscal-report

<table>
<thead>
<tr>
<th>accelerated</th>
<th>adjusted</th>
<th>advance</th>
<th>amortization</th>
<th>anniversary</th>
</tr>
</thead>
<tbody>
<tr>
<td>annual</td>
<td>annualized</td>
<td>beats</td>
<td>capping</td>
<td>chinabondcomcn</td>
</tr>
<tr>
<td>chinamoneycomcn</td>
<td>companywide</td>
<td>consecutive</td>
<td>csoi</td>
<td>decline</td>
</tr>
<tr>
<td>declines</td>
<td>decrease</td>
<td>depreciation</td>
<td>distributable</td>
<td>drop</td>
</tr>
<tr>
<td>during</td>
<td>earnings</td>
<td>ebitda</td>
<td>estimated</td>
<td>ebitda</td>
</tr>
<tr>
<td>eighth</td>
<td>erased</td>
<td>erasing</td>
<td>exceeding</td>
<td>first</td>
</tr>
<tr>
<td>extended</td>
<td>extending</td>
<td>fifth</td>
<td>first</td>
<td>first</td>
</tr>
<tr>
<td>following</td>
<td>fourth</td>
<td>gain</td>
<td>impairments</td>
<td>its</td>
</tr>
<tr>
<td>jump</td>
<td>krooni</td>
<td>litai</td>
<td>longest</td>
<td>loss</td>
</tr>
<tr>
<td>losses</td>
<td>losshtl</td>
<td>margins</td>
<td>marked</td>
<td>marking</td>
</tr>
<tr>
<td>matching</td>
<td>milion</td>
<td>misses</td>
<td>monthly</td>
<td>net</td>
</tr>
<tr>
<td>ninth</td>
<td>operating</td>
<td>pared</td>
<td>pares</td>
<td>paring</td>
</tr>
<tr>
<td>plunge</td>
<td>posted</td>
<td>posting</td>
<td>posts</td>
<td>pretax</td>
</tr>
<tr>
<td>profit</td>
<td>profits</td>
<td>pula</td>
<td>purchases</td>
<td>quadruples</td>
</tr>
<tr>
<td>quarter</td>
<td>quarterly</td>
<td>rallies</td>
<td>rally</td>
<td>reported</td>
</tr>
<tr>
<td>restated</td>
<td>retreat</td>
<td>revenue</td>
<td>reversing</td>
<td>saartotl</td>
</tr>
<tr>
<td>second</td>
<td>seventh</td>
<td>sixth</td>
<td>slide</td>
<td>slump</td>
</tr>
<tr>
<td>snapping</td>
<td>steepest</td>
<td>straight</td>
<td>third</td>
<td>thirteenth</td>
</tr>
<tr>
<td>totalled</td>
<td>trails</td>
<td>trimming</td>
<td>triples</td>
<td>unadjusted</td>
</tr>
<tr>
<td>unaudited</td>
<td>undistributed</td>
<td>unrepeated</td>
<td>website</td>
<td>weekly</td>
</tr>
<tr>
<td>writedown</td>
<td>yearly</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### B.7 Category: investment

<table>
<thead>
<tr>
<th>accumulated</th>
<th>acquire</th>
<th>acquired</th>
<th>acquires</th>
<th>acquiring</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>allocate</td>
<td>allocated</td>
<td>allocates</td>
<td>allocating</td>
</tr>
<tr>
<td>allocation</td>
<td>allocations</td>
<td>allotted</td>
<td>amass</td>
<td>amassed</td>
</tr>
<tr>
<td>amassing</td>
<td>assets</td>
<td>assist</td>
<td>attract</td>
<td>attracted</td>
</tr>
<tr>
<td>attracting</td>
<td>birr</td>
<td>borrow</td>
<td>borrowed</td>
<td>bought</td>
</tr>
<tr>
<td>budgeted</td>
<td>build</td>
<td>buy</td>
<td>buys</td>
<td>cater</td>
</tr>
<tr>
<td>combine</td>
<td>complement</td>
<td>concentrated</td>
<td>consume</td>
<td>create</td>
</tr>
<tr>
<td>deploy</td>
<td>develop</td>
<td>devote</td>
<td>disbursed</td>
<td>dispose</td>
</tr>
<tr>
<td>distribute</td>
<td>divest</td>
<td>earmarked</td>
<td>earn</td>
<td>earned</td>
</tr>
<tr>
<td>encourage</td>
<td>entice</td>
<td>facilities</td>
<td>financings</td>
<td>funds</td>
</tr>
<tr>
<td>generate</td>
<td>handle</td>
<td>holds</td>
<td>install</td>
<td>interested</td>
</tr>
<tr>
<td>Action Verbs</td>
<td>Action Verbs</td>
<td>Action Verbs</td>
<td>Action Verbs</td>
<td>Action Verbs</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>invest</td>
<td>invested</td>
<td>investing</td>
<td>investment</td>
<td>investments</td>
</tr>
<tr>
<td>invests</td>
<td>leftover</td>
<td>loaned</td>
<td>lure</td>
<td>luring</td>
</tr>
<tr>
<td>managed</td>
<td>manages</td>
<td>merge</td>
<td>netted</td>
<td>offer</td>
</tr>
<tr>
<td>offload</td>
<td>operate</td>
<td>oversees</td>
<td>owns</td>
<td>partners</td>
</tr>
<tr>
<td>portfolio</td>
<td>portfolios</td>
<td>poured</td>
<td>prospered</td>
<td>purchase</td>
</tr>
<tr>
<td>purchased</td>
<td>racked</td>
<td>raise</td>
<td>recycle</td>
<td>redeem</td>
</tr>
<tr>
<td>refurbish</td>
<td>repay</td>
<td>save</td>
<td>sell</td>
<td>sells</td>
</tr>
<tr>
<td>sold</td>
<td>specializes</td>
<td>spend</td>
<td>spends</td>
<td>spent</td>
</tr>
<tr>
<td>stakes</td>
<td>swapped</td>
<td>tap</td>
<td>unload</td>
<td>utilize</td>
</tr>
</tbody>
</table>

### B.8 Category: bankrupt

<table>
<thead>
<tr>
<th>Company/Word</th>
<th>Company/Word</th>
<th>Company/Word</th>
<th>Company/Word</th>
<th>Company/Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>abitibibowater</td>
<td>accentia</td>
<td>acln</td>
<td>adjudicating</td>
<td>adjudication</td>
</tr>
<tr>
<td>allowing</td>
<td>ambac</td>
<td>americanwest</td>
<td>arcapita</td>
<td>atterbury</td>
</tr>
<tr>
<td>awal</td>
<td>bankrupt</td>
<td>bankruptcy</td>
<td>banning</td>
<td>berkline</td>
</tr>
<tr>
<td>bicent</td>
<td>briefs</td>
<td>broadlane</td>
<td>broadsign</td>
<td>capmark</td>
</tr>
<tr>
<td>chapter</td>
<td>choicepoint</td>
<td>cit</td>
<td>clarendon</td>
<td>clarita</td>
</tr>
<tr>
<td>clearlake</td>
<td>cocopah</td>
<td>complaint</td>
<td>conservation</td>
<td>consummates</td>
</tr>
<tr>
<td>dbsi</td>
<td>dynegy</td>
<td>earthrenew</td>
<td>ebg</td>
<td>enforcement</td>
</tr>
<tr>
<td>enforcing</td>
<td>environmental</td>
<td>eyecare</td>
<td>filed</td>
<td>forbids</td>
</tr>
<tr>
<td>frivolous</td>
<td>giddens</td>
<td>gsc</td>
<td>gtech</td>
<td>hpht</td>
</tr>
<tr>
<td>hydrl</td>
<td>insolvency</td>
<td>involuntary</td>
<td>laws</td>
<td>lawsuit</td>
</tr>
<tr>
<td>liquidating</td>
<td>liquidation</td>
<td>liquidator</td>
<td>lodged</td>
<td>madoff</td>
</tr>
<tr>
<td>mandating</td>
<td>maxam</td>
<td>mechanisms</td>
<td>motions</td>
<td>ordinances</td>
</tr>
<tr>
<td>outlawing</td>
<td>outlaws</td>
<td>palmdale</td>
<td>petition</td>
<td>petitioners</td>
</tr>
<tr>
<td>petitions</td>
<td>picard</td>
<td>pittsburg</td>
<td>prepackaged</td>
<td>procedures</td>
</tr>
<tr>
<td>prohibit</td>
<td>prohibiting</td>
<td>prohibits</td>
<td>protection</td>
<td>protections</td>
</tr>
<tr>
<td>protocols</td>
<td>qvt</td>
<td>regulations</td>
<td>reorganization</td>
<td>required</td>
</tr>
<tr>
<td>requires</td>
<td>requiring</td>
<td>restructuring</td>
<td>roomstore</td>
<td>safeguards</td>
</tr>
<tr>
<td>safety</td>
<td>sanctioning</td>
<td>shoppes</td>
<td>statutes</td>
<td>townends</td>
</tr>
<tr>
<td>tremont</td>
<td>tridimension</td>
<td>tronox</td>
<td>trustee</td>
<td>unconstitutionally</td>
</tr>
<tr>
<td>unlawful</td>
<td>unsealed</td>
<td>vertis</td>
<td>zeneca</td>
<td>zoning</td>
</tr>
</tbody>
</table>

### B.9 Category: government

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abhisit</td>
<td>admin</td>
<td>agcy</td>
<td>agency</td>
<td>austerityhtml</td>
</tr>
<tr>
<td>bailouthtml</td>
<td>ballot</td>
<td>barack</td>
<td>biacora</td>
<td>boiko</td>
</tr>
<tr>
<td>borisssov</td>
<td>bouh</td>
<td>bush</td>
<td>cash</td>
<td>clinton</td>
</tr>
<tr>
<td>congress</td>
<td>deductions</td>
<td>dejan</td>
<td>directorate</td>
<td>discord</td>
</tr>
<tr>
<td>donilon</td>
<td>electable</td>
<td>election</td>
<td>elections</td>
<td>elshad</td>
</tr>
<tr>
<td>geithner</td>
<td>geospatial</td>
<td>goverment</td>
<td>government</td>
<td>governor</td>
</tr>
</tbody>
</table>