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

Yangtuo 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

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

List of Tables

List of Figures

Abbreviations

DNN	**D**eep **N**eural **N**etwork
NLP	**N**ature **L**anguage **P**rocessing
SGD	**S**tochastic **G**radient **D**escent
BoW	**B**ag **o**f **W**ords
EMA	**E**xponential **M**oving **A**verage
RSI	**R**elative **S**trength **I**ndex
TF-IDF	**T**erm **F**requency-**I**nverse **D**ocument **F**requency
PMI	**P**ointwise **M**utual **I**nformation

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.

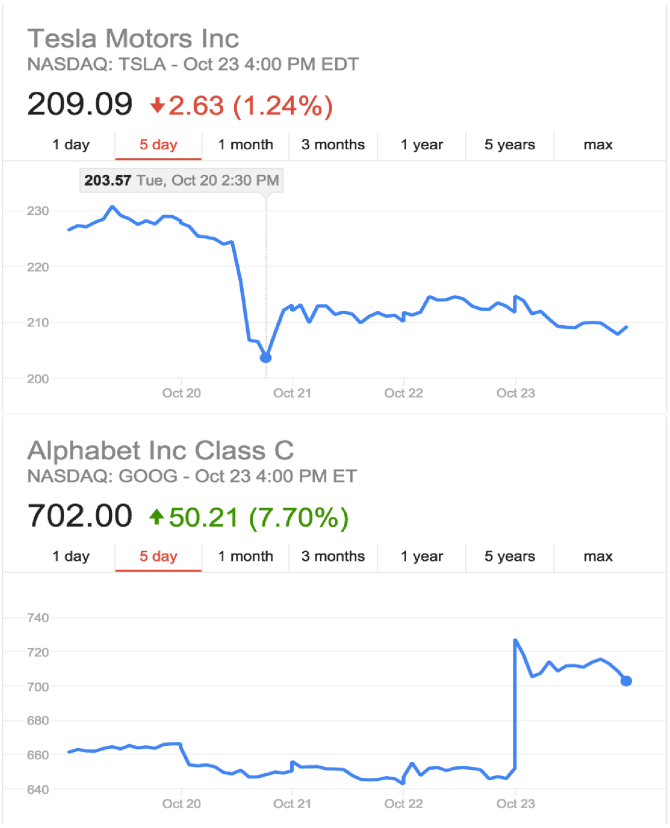

FIGURE 1.1: Illustration of the volatility of stock market.

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 then 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 summarizing 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 [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

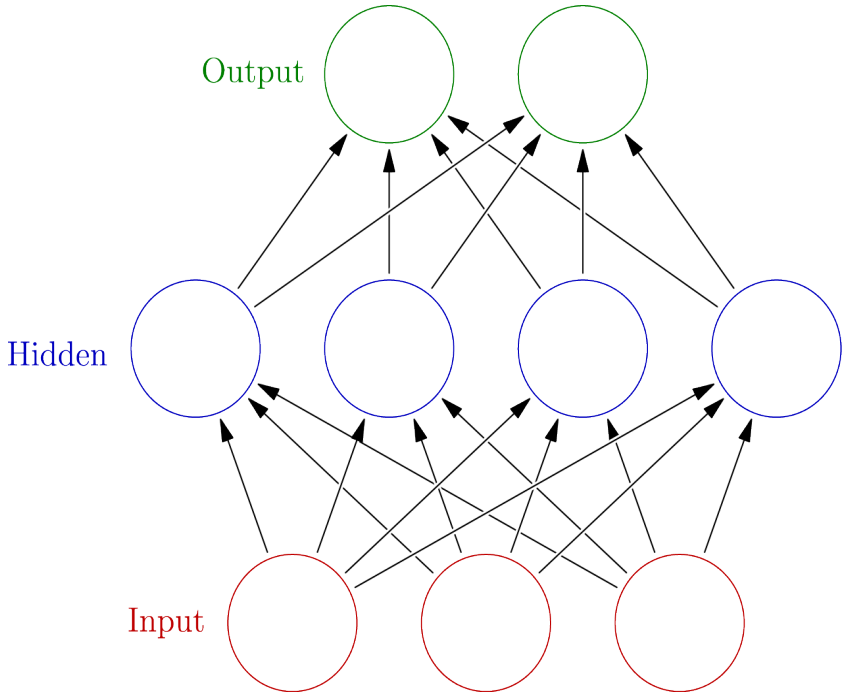

FIGURE 2.1: An simple example of neural network with a single hidden layer. Each circle represents a computing unit in the network. Each arrow represents the connection between two units. The direction of arrow indicates that the input information is propagated from input layer at the bottom to the output 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, $\mathbf{x} = [x_1, x_2, ..., x_N]$, 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 $\mathbf{x} = \{x_1, x_2, ..., x_N\}$ 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 *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 *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 *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 *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:

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

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:

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

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 $\underline{\mathrm{W}}$ 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:

$$ReLU(a) = max(0, a)$$

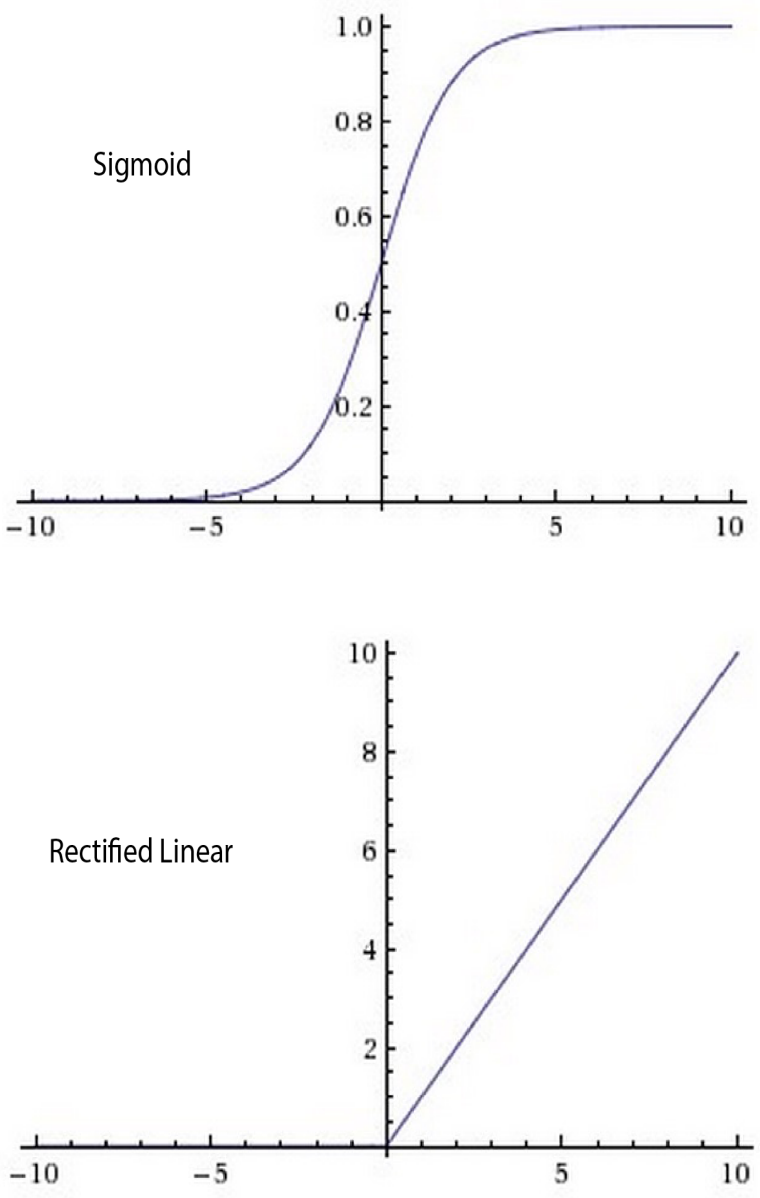

FIGURE 2.2: Two examples of activation functions.

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 dealing with. If the output has k dimensions, where $k > 1$, the network can be use 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 *stock-up* and *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 $loss(\mathbf{y}, \mathbf{y}')$, takes the input $\mathbf{y}$ which is network's output vector contains predictive value for each output unit and $\mathbf{y}'$ is the vector stores actual *correct* values of the corresponding input samples. Such loss function will output the loss of predicting $\mathbf{y}$ when the correct output should be $\mathbf{y}'$. 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:

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

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:

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

where N is the total number of available classes. $\mathbf{y}'$ 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. $\mathbf{y}$ 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 $\mathbf{y}$ is differ from the expected distribution $\mathbf{y}'$.

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 $\mathbf{y} = \{0.1, 0.9\}$, 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 $\mathbf{y}' = \{1.0, 0.0\}$. So the loss function is $loss(\mathbf{y}, \mathbf{y}') = -\sum_{i=1}^{2} y_i' \log(y_i)$. Because $\mathbf{y}$ is the output of the network, $\mathbf{y} = NN(\mathbf{x}, \mathbf{W})$. Therefore the training algorithm's goal is to adjust the weight matrix $\mathbf{W}$ subject to minimize $loss_{cross-entropy}$.

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 $f(\mathbf{x}, \theta)$, where θ is the adjustable parameter of f; a training dataset includes the input samples $\mathbf{X} = \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$ and the corresponding outputs, or *labels*, $\mathbf{Y} = \mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_n}$; 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 $\mathbf{W}$. The goal of SGD is to repeatedly set the parameter θ 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 $\{\mathbf{x_i}, \mathbf{y_i}\}$ from the training dataset, and feed $\mathbf{x_i}$ to function f. Then it computes the value of loss function by $loss(f(x_i, \theta), y_i)$, and get the gradient of the loss function with respect to the parameter θ. Finally, it updates the parameter using the gradient information in the following way:

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

where $\nabla loss(f(x_i, \theta), y_i)$ denotes the gradient of loss function, and η 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

θ 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 a important attribute in the algorithm that should be set carefully because if we set it too big, the pace of adjusting θ 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 θ to θ', we first feed the inputs from validation dataset to the network with updated parameter θ', 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 θ and missed the local minimum, we reduce the learning rate with certain ratio and revert the parameter back to θ. 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 alway 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 θ:

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

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 a 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, ie. $\frac{\partial L}{\partial \mathbf{W}}$, 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:

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

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:

$$a_j = \sum_i w_{ji} x_i \tag{2.1}$$

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:

$$z_j = g(a_j) \tag{2.2}$$

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}:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{2.3}$$

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

$$\frac{\partial a_j}{\partial w_{ji}} = x_i \tag{2.4}$$

In addition, we denote the term:

$$\delta_j \equiv \frac{\partial L}{\partial a_j} \tag{2.5}$$

where δ 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, δ_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:

$$\frac{\partial L}{\partial w_{ji}} = \delta_j x_i \tag{2.6}$$

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 δ_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 δ_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):

$$\delta_j \equiv \frac{\partial L}{\partial a_j} = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.7}$$

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:

$$z_j = g(a_j) \tag{2.8}$$

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 ∂a_k over ∂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*:

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \tag{2.9}$$

which evolves evaluation of two simpler derivatives, $g'(a_j)$ is just the derivative of the activation function used for all hidden units; and δ_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 δ 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 δ_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 δ 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 δ_k. In the backpropagate stage, it backpropagates the δ's, starting from δ_k, to get all errors for every hidden unit in the network. At last, the algorithm inserts δ's back to equation (2.6) to evaluate the required gradients.

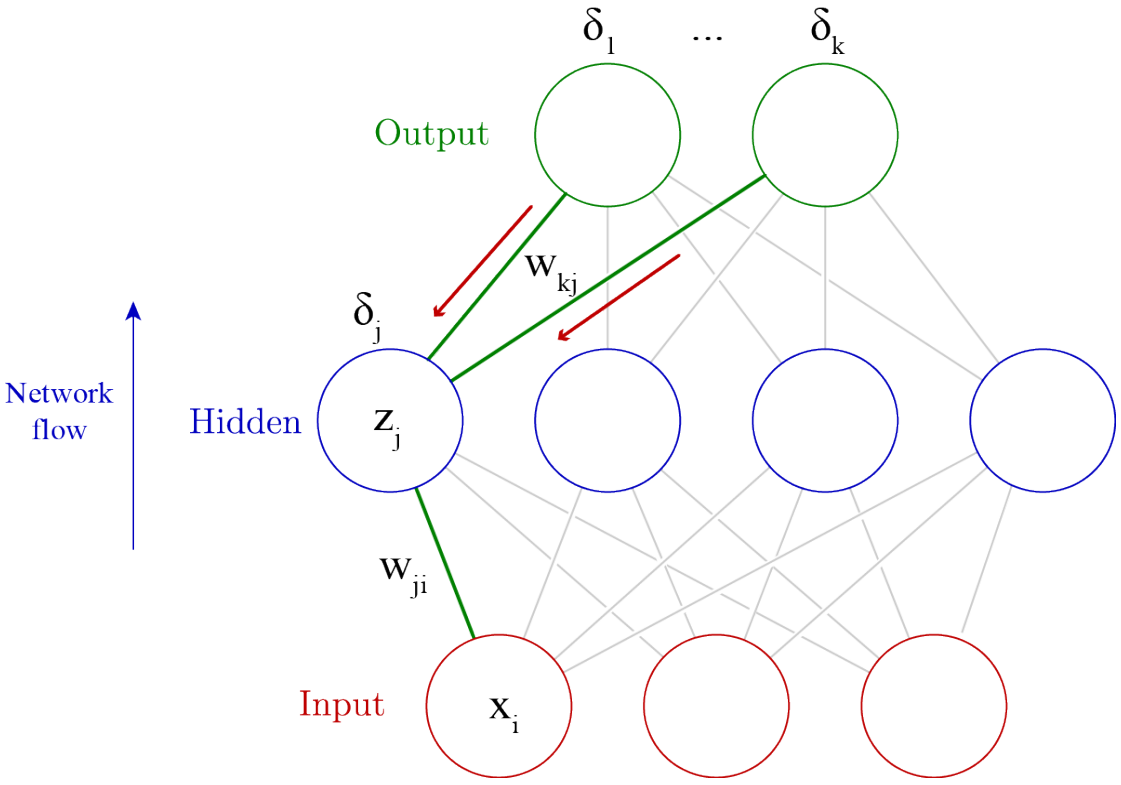

FIGURE 2.3: Overall procedure of error back propagation algorithm. Blue arrow indicates the direction of feed forward networks flow from input layer to output layer. Red arrows indicates the direction of backward error propagation.

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 language 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:

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

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: $P(w_i|context)$, 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:

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

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:

$$C(\text{“}queen\text{”}) - C(\text{“}women\text{”}) + C(\text{“}man\text{”}) \approx C(\text{“}king\text{”})$$

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 representations 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

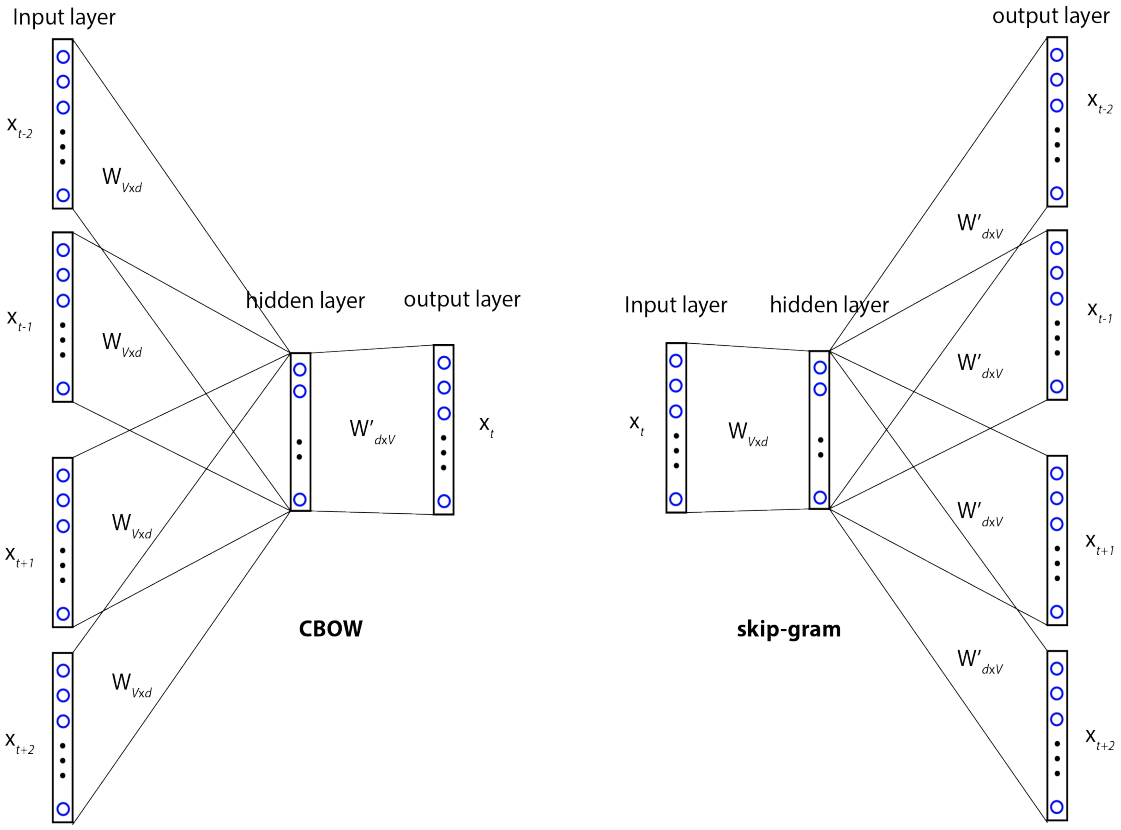

FIGURE 3.1: Architecture of two models for learning high quality word embedding proposed in [1]. Both models use the word sequece $\{x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}\}$ as an example. CBOW takes $\{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\}$ as input and predict the middle word x_t. Whereas skip-gram 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:

$$\log p(x_t|context)$$

where *context* is the surrounding words $\{x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\}$, and $p(x_t|context)$ 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 *1 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 Extimation* (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:

$$\log \sigma({v'_{x_{t+j}}}^\top v_{x_t}) + \sum_{i=1}^{k} E_{x_i} \sim P_n(x)[\log \sigma(-{v'_{x_i}}^\top v_{x_i})]$$

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 $3/4rd$ power of the unigram distribution of x gives the optimal result:

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

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 takes 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

Field name	**Description**	**Example**
PERMNO	Stock identifier, CRSP's unique and permanent issue identification number.	90319
Date	Date of the record with format "YYYY-MM-DD".	2006-01-03
Company name	Full name of the company	GOOGLE INC
Close price	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.	435.23
Open price	Open price on that trading day.	422.52
Ask high	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.	435.67
Bid low	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.	418.22
Number of trades	Total number of trades made during the day, set to 99 if not available.	70118
Holding period return	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.	0.04910
Market value weighted return	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.	0.01641
Market equal weighted return	Similar with Market value weighted return except the market index is computed base on equal weighted stock value in the market	0.01098
S&P index return	Return on the Standard & Poor's Composite index.	0.01643

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:

$F = \{$*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:

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

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

$$\mathbf{F} = (F_{closeprice}, F_{openprice}, \ldots, F_{S\&Pindexreturn})$$

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. α 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:

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

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

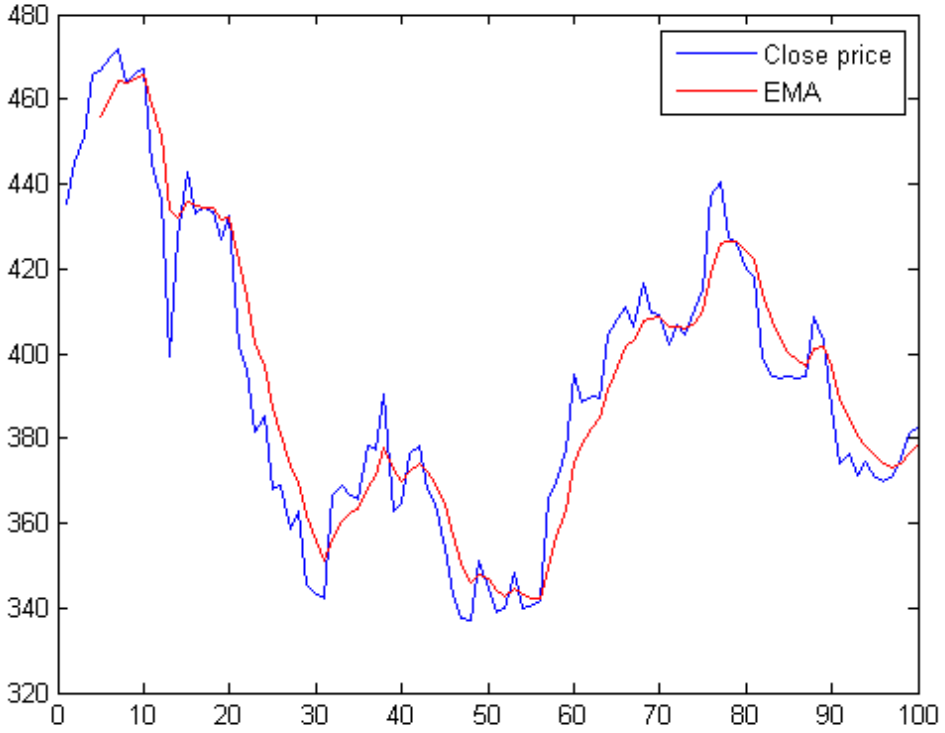

FIGURE 5.1: Example of exponential moving average of Google's close price during 2006-01-03 and 2006-05-25. The fluctuation in original data is smooth out in EMA.

calculate the *upward change* U and *downward change* D:

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

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

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

$$RSI = 100 - \frac{100}{1 + \frac{EMA(U)}{EMA(D)}}$$

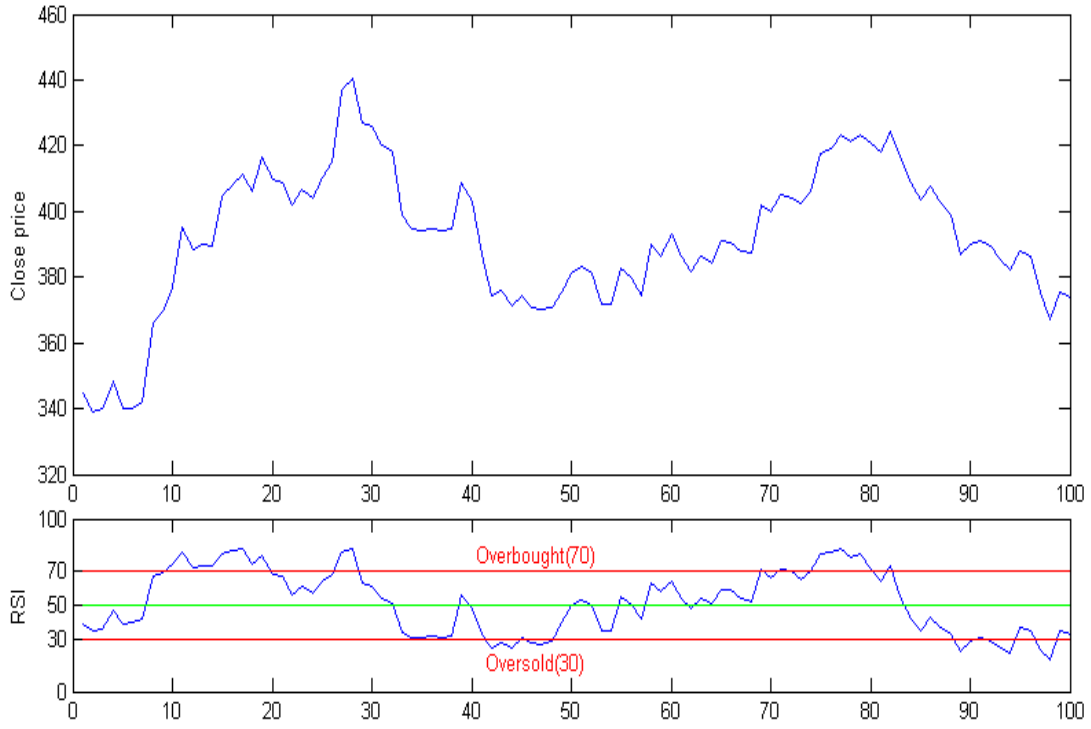

FIGURE 5.2: Example of relative strength index of Google's close price. The region between 30 and 70 is typically the neutral region. RSI higher then 70 indicates an *overbought* of the stock and price drop is about to happen, whereas RSI lower then 30 indicates an *oversold* of the stock and price raise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

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:

$$\Delta F_{closeprice} = (P_{t-4}, P_{t-3}, P_{t-2}, P_{t-1}) - (P_{t-5}, P_{t-4}, P_{t-3}, P_{t-2})$$

And the second order difference is calculated by taking the difference between two adjacent values in each $\Delta F_{closeprice}$:

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

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

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

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 suffices 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

Field name	Example
Date	2006-11-01
Title	Apple pushes Oricon to quit PC music downloads
Body	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 ...

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 base 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:

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

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 *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:

$$\text{TF-IDF}(w, s, S) = TF(w, s) * IDF(w, S)$$

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:

$$\text{PMI}(w, pos) = \log \frac{\frac{\text{freq}(w,pos)}{N}}{\frac{\text{freq}(w)}{N} \times \frac{\text{freq}(pos)}{N}}$$

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 $\frac{\text{freq}(w,pos)}{N}$ 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 $\frac{\text{freq}(w)}{N}$ denotes the probability of keyword being w; and freq(pos) denotes the total number of positive samples in the training set, and $\frac{\text{freq}(pos)}{N}$ denotes the probability of an article being labeled as *positive*. By cancelling out the $N's$, the formula can be simplified to:

$$\text{PMI}(w, pos) = \log \frac{\text{freq}(w, pos) \times N}{\text{freq}(w) \times \text{freq}(pos)}$$

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

$$\text{PMI}(w, nag) = \log \frac{\text{freq}(w, nag) \times N}{\text{freq}(w) \times \text{freq}(nag)}$$

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

$$\text{PS}(w) = \text{PMI}(w, pos) - \text{PMI}(w, neg).$$

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

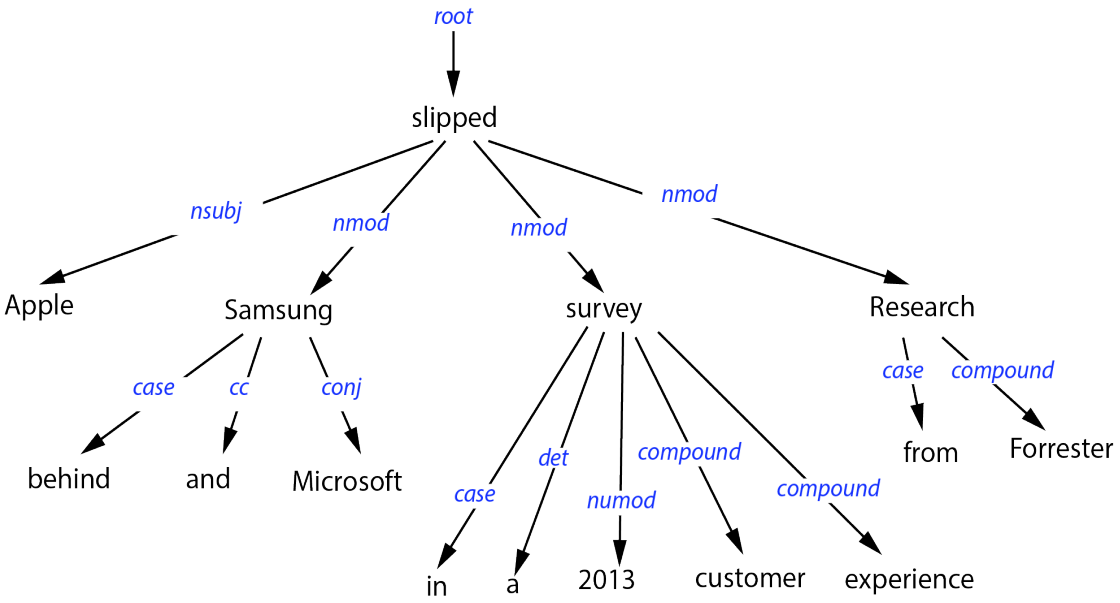

FIGURE 6.1: Parsing result of the example *"Apple slipped behind Samsung and Microsoft in a 2013 customer experience survey from Forrester Research"*.

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.

Date	*News Title*
2007-01-09	Investors dump RIM as Apple launches iPhone.
2007-01-09	FACTBOX-Key facts on Apple's new iPhone.
2007-01-09	Apple introduces svelte multimedia iPhone.
2007-01-09	Apple unveils iPhone.
2007-01-09	Apple rolls out much-anticipated iPhone.
2007-01-09	Apple shares up after Jobs introduces mobile phone.

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

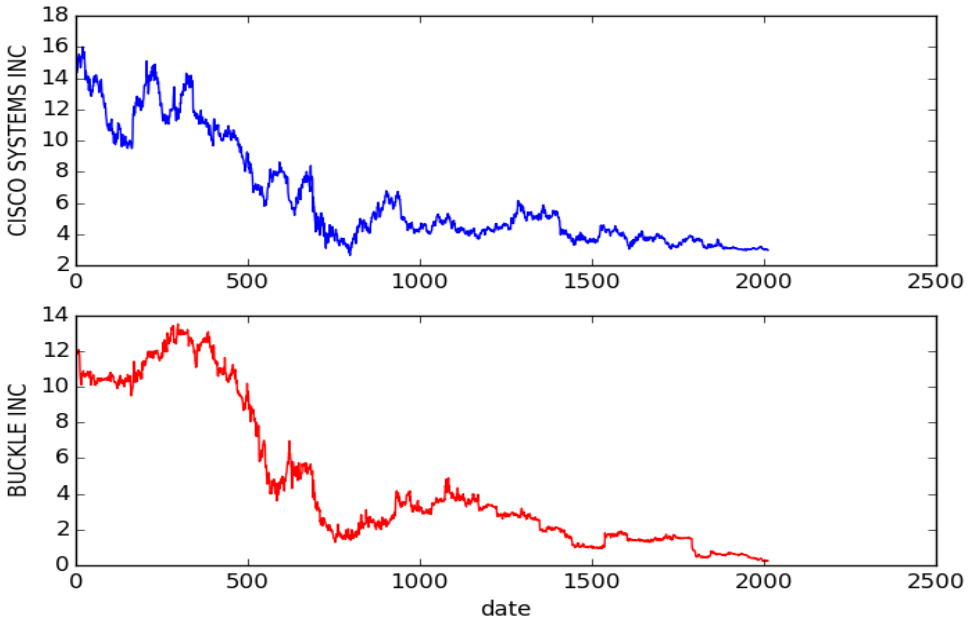

FIGURE 7.1: An example shows a pair of stocks which are highly related to each other. The two graphs above plot the closing price of "CISCO SYSTEM INC" and "BUCKLE INC" during the period of 2006-01-03 to 2013-12-31.

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 stock 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.

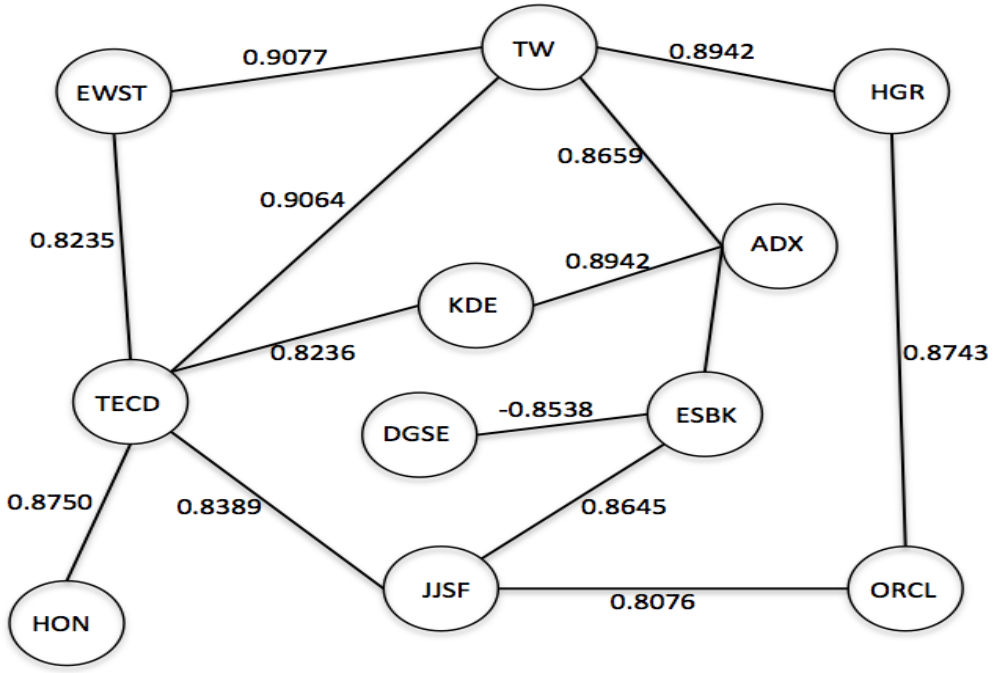

FIGURE 7.2: Illustration of a part of correlation graph which contains 11 stocks. The symbol in the circle are the ticker name of the stock. The value along the edges are the correlation score of the two stocks connected by that edge.

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:

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

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; $\mathbf{E}[A]$ denotes the *expected value*(or average value) of the close price of stock A. Then we calculate the correlation coefficient by:

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

Where $cov(A, A)$ and $cov(B, B)$ 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:

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

where n is the number of times the signal got propagated through the matrix. The prediction $\mathbf{x^{(n)}}$ 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 base 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 base 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.

input timeframe	*DNN structure*	*learning rate*	*mini-batch size*	*error rate*
20 days	[1024 1024 1024]	0.005	200	50.09%
14 days	[1024 1024 1024]	0.005	200	48.96%
7 days	[1024 1024 1024]	0.005	200	48.23%
5 days	[1024 1024 1024]	0.005	200	**47.64%**
4 days	[1024 1024 1024]	0.005	200	48.09%
3 days	[1024 1024 1024]	0.005	200	48.45%
2 days	[1024 1024 1024]	0.005	200	50.25%

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

input timeframe	*DNN structure*	*learning rate*	*mini-batch size*	*error rate*
5 days	[500]	0.005	200	50.00%
5 days	[1024]	0.005	200	50.01%
5 days	[1024 1024]	0.005	200	48.79%
5 days	**[1024 1024 1024]**	0.005	200	**47.64%**
5 days	[1024 1024 1024 1024]	0.005	200	47.64%
5 days	[1024 1024 1024 1024 1024]	0.005	200	47.62%

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 generates almost the same performance, we choose the structure of 3 layers as the optimal one as it has the least computational cost.

input timeframe	*DNN structure*	*learning rate*	*mini-batch size*	*error rate*
5 days	[1024 1024 1024]	0.01	200	49.89%
5 days	[1024 1024 1024]	**0.005**	200	**47.64%**
5 days	[1024 1024 1024]	0.001	200	47.64%
5 days	[1024 1024 1024]	0.0008	200	48.00%

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.

input timeframe	*DNN structure*	*learning rate*	*mini-batch size*	*error rate*
5 days	[1024 1024 1024]	0.005	100	48.89%
5 days	[1024 1024 1024]	0.005	**200**	**47.64%**
5 days	[1024 1024 1024]	0.005	1000	47.92%

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.

feature combination	*error rate*
random guess	50.23%
price	47.64%
price + BoK	46.02%
price + BoK + PS	43.96%
price + BOK + CT	45.86%
price + PS	45.00%
price + CT	46.10%
price + PS +CT	46.03%
price + BoK + PS + CT	**43.13%**
structured events [27]	44.79%

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* stats 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:

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

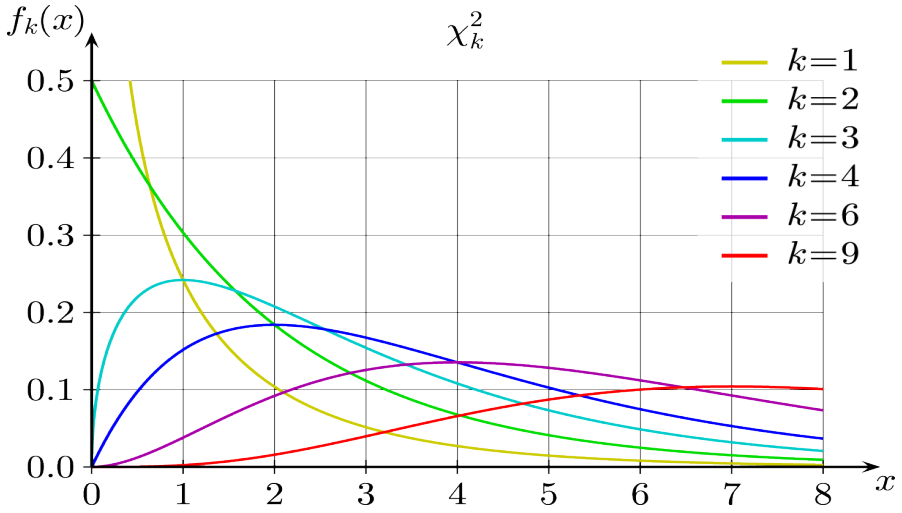

FIGURE 8.1: Probability density function of Chi-squared distribution. k denotes the degree of freedom.

with the assumption that the null hypothesis is true. χ^2 has a *chi-squared distribution* (shown in figure 8.1) with 1 degree of freedom. If χ^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 χ^2 is measured by a *p-value* with respect to a significance level α (traditionally set to 0.05 or 0.001). In our case, the *p-value* is the probability of the variable greater then the computed χ^2 in the chi-square distribution, i.e. $P(x > \chi^2)$.

	Test 2 positive	Test 2 negative	Row total
Test 1 positive	a	b	a + b
Test 1 negative	c	d	c + d
Column total	a + c	b + d	n

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 χ^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 χ^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 α value of 0.001, which provides a strong evidence to reject the null hypothesis of random guess.

	DNN w/ price correct	DNN w/ price wrong	Row total
Random guess correct	2,860	2,609	5,469
Random guess wrong	2,853	2,589	5,480
Column total	5,713	5,198	10,911

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

feature combination	*error rate*	χ^2	*p-value*
random guess	50.23%	n/a	n/a
price	47.64%	10.90	0.00096
price + BoK	46.02%	36.09	1.88×10^{-9}
price + BoK + PS	43.96%	78.81	6.83×10^{-19}
price + BOK + CT	45.86%	36.09	1.88×10^{-9}
price + PS	45.00%	55.19	1.09×10^{-13}
price + CT	46.10%	38.94	4.37×10^{-10}
price + PS +CT	46.03%	36.09	1.88×10^{-9}
price + BoK + PS + CT	43.13%	99.52	1.94×10^{-23}
structured events [27]	44.79%	60.70	6.64×10^{-15}

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

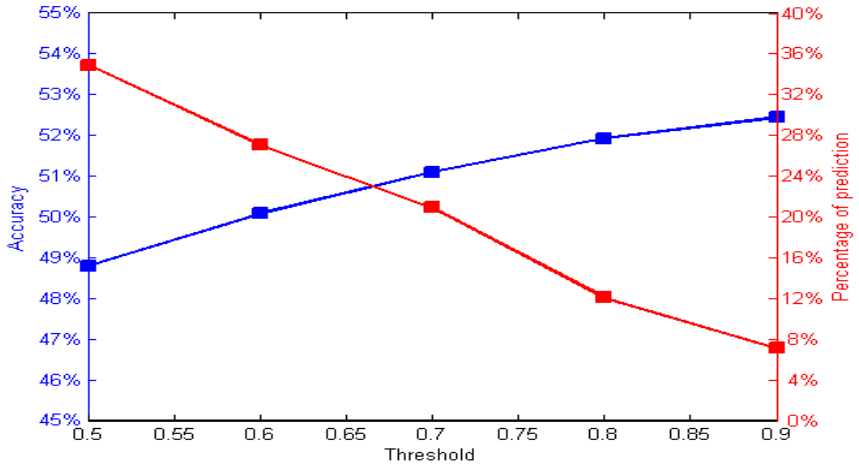

FIGURE 8.2: Predict unseen stocks via correlation

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 new 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

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*, 2013.

[2] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech 2011*. International Speech Communication Association, August 2011. URL http://research.microsoft.com/apps/pubs/default.aspx?id=153169.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[4] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1082.

[5] Wenzhe Pei, Tao Ge, and Baobao Chang. An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 313–322, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1031.

[6] Greg Durrett and Dan Klein. Neural crf parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1030.

[7] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1032.

[8] Balázs Csanád Csáji. Approximation with artificial neural networks. Master's thesis, Faculty of Sciences, Etvs Lornd University, Hungary, 2001.

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[10] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[11] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[12] Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006.

[13] Iebeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10:215–236, 1991.

[14] Monica Adya and Fred Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *Journal of Forecasting*, 17:481–495, 1991.

[15] Man-Chung Chan, Chi-Cheong Wong, and Chi-Chung Lam. Financial time series forecasting by neural network using conjugate gradient learning algorithm and multiple linear regression weight initialization. *Computing in Economics and Finance*, 61, 2000.

[16] Andrew Skabar and Ian Cloete. Neural networks, financial trading and the efficient markets hypothesis. In *Proc. the Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002.

[17] Xiaotian Zhu, Hong Wang, Li Xu, and Huaizu Li. Predicting stock index increments by neural networks: The role of trading volume under different horizons. *Expert Systems with Applications*, 34:30433054, 2008.

[18] Heeyoung Lee, Mihai Surdeanu, Bill MacCartney, and Dan Jurafsky. On the importance of text analysis for stock price prediction. 2014.

[19] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[20] J.J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance Series. New York Institute of Finance, 1999. ISBN 9780735200661. URL https://books.google.ca/books?id=5zhXEqdr_IcC.

[21] Suraphan Thawornwong and David Enke. Forecasting stock returns with artificial neural networks. *Neural Networks in Business Forecasting*, pages 47–79, 2003.

[22] Chicago Booth. *CRSP Data Description Guide for the CRSP US Stock Database and CRSP US Indices Database*. Center for Research in Security Prices, The University of Chicago Graduate School of Business (https://wrds-web.wharton.upenn.edu/wrds/index.cfm), 2012.

[23] J.W. Wilder. *New Concepts in Technical Trading Systems*. Trend Research, 1978. ISBN 9780894590276. URL https://books.google.ca/books?id=WesJAQAAMAAJ.

[24] Boyi Xie, Rebecca J. Passonneau, Leon Wu, and Germán G. Creamer. Semantic frames to predict stock price movement. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–883, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-1086.

[25] Jianfeng Si, Arjun Mukherjee, Bing Liu, Qing Li, Huayi Li, and Xiaotie Deng. Exploiting topic based twitter sentiment for stock prediction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 24–29, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-2005.

[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*

2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1139–1145, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1120.

[27] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Using structured events to predict stock price movement: An empirical investigation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1415–1425, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1148.

[28] Roy Bar-Haim, Elad Dinur, Ronen Feldman, Moshe Fresko, and Guy Goldstein. Identifying and following expert investors in stock microblogs. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1310–1319, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D11-1121.

[29] Marie-Catherine Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings LREC*, 2006.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, pages 3111–3119, 2013.

[31] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361, 2012.

[32] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010,*

11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010, pages 1045–1048, 2010.

[33] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January 2010. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1861751.1861756.

[34] Peter D. Turney and Michael L. Littman. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Trans. Inf. Syst.*, 21(4):315–346, October 2003. ISSN 1046-8188. doi: 10.1145/944012.944013. URL http://doi.acm.org/10.1145/944012.944013.

[35] D. Okanohara and J. Tsujii. Assigning polarity scores to reviews using machine learning techniques. In R. Dale, K. F. Wong, J. Su, and O. Y. Kwong, editors, *Natural Language Processing - IJCNLP 2005*, volume 3651 of *Lecture Notes in Computer Science*. Springer-Verlag, Jeju Island, Korea, October 2005.

[36] B.G. Malkiel. *A Random Walk Down Wall Street: Including a Life-cycle Guide to Personal Investing*. Norton, 1999. ISBN 9780393320404. URL https://books.google.ca/books?id=fAsZGQfmXG8C.

[37] Jia Pan, Cong Liu, Zhiguo Wang, Yu Hu, and Hui Jiang. Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling. In *Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on*, pages 301–305, Dec 2012. doi: 10.1109/ISCSLP.2012.6423452.

[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

Proc. of the 53th Annual Meeting of the Association for Computational Linguistics (ACL 2015), page 495, July 2015.

[39] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(10):1533–1545, 2014.

[40] Yebo Bao, Hui Jiang, Cong Liu, Yu Hu, and Lirong Dai. Investigation on dimensionality reduction of concatenated features with deep neural network for lvcsr systems. In *Signal Processing (ICSP), 2012 IEEE 11th International Conference on*, volume 1, pages 562–566. IEEE, 2012.

[41] Zhigang Chen, Wei Lin, Qian Chen, Xiaoping Chen, Si Wei, Hui Jiang, and Xiaodan Zhu. Revisiting word embedding for contrasting meaning. In *Proc. of the 53th Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, July 2015.

[42] Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proc. of the 53th Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, July 2015.

[43] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, 2013. ISBN 9788132209065. URL https://books.google.ca/books?id=HL4HrgEACAAJ.

[44] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[45] Yangtuo Peng and Hui Jiang. Leverage financial news to predict stock price movements using word embeddings and deep neural networks. In *Proc. of the The 15th*

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

Appendix A

Bag of Keywords Features

abandon	abates	abating	absence	accelerate
accelerated	accelerates	accelerateshtml	accelerating	acceleration
accelerations	accumulated	achieve	add	adjust
advance	advanced	advances	advanceshtml	advancing
affect	affection	aftermath	albeit	aldershot
allaying	amid	amidst	anaemic	anchoring
anemic	anomalies	appreciable	appreciate	appreciated
appreciates	appreciating	appreciation	approximate	automatic
awhile	backdrop	backlash	backstop	backwash
balloon	ballooned	ballooning	bask	battering
batters	bear	bearishness	beat	beating
beats	blanket	bleak	blip	blossomed
bolster	bolstering	boost	boosting	bounceback
breakout	bring	budget	bull	bullishly
buoy	buoyant	buoying	buoys	calamity
calibrating	calibration	calming	calms	carve
catalyst	cause	caution	cautiousness	cbo
centerpiece	centimes	challenges	changes	chaos
cheer	chokes	choppy	climb	climbed
climbing	climbs	clinch	cloth	clouded
cncpiyoy	come	comeback	complacency	concernhtml
conditions	conflagration	conflicting	confounding	consequent
considerably	constrain	constraint	constrict	context
continuation	contracted	contracting	contraction	contractions
contrasting	contribute	converge	converging	convulsing
cooldown	cooled	cooling	cools	coolshtml
correction	countering	cratering	credits	crimp
crimped	crimping	crimps	crippling	crises

crisis cspe culmination cumulative curb
curbed curbing curtail curtailed cut
cutback cutbacks cuts cycle damp
damped dampen dampened dampener damping
damps danger darkening debase debasement
debilitating decarbonization decelerate decelerated decelerates
decelerating deceleration decent decline declinehtml
declines declineshtml declining decrease decreased
decreases decreasing deepened deepening deepens
defend deficit deficithtml deficits deflate
deluge dent dented denting dents
depreciate depreciated depreciates depreciating depress
depressed depressing derail derailing derecho
deservedly destocking deteriorating deterioration dethrone
devaluation devalue devalued devaluing differ
differing difficulties dig diminished diminution
dimmer dimming dims dip dipped
disappear disappointment discernible discourage disincentive
disinflation disinflationary dislocation dislocations displace
disruption disruptions dissipate distortion distortions
diverge diverged divergent diverging dnipro
dollar dollarhtml downdraft downdrafts downshift
downtrend downturn downturns draconian dramatic
drift drifts drive drop drophtml
dropped dropping drops droughts durables
dwindling dynamism earningshtml ease eased
eases easeshtml ebbed ebbing ebbs
eclipsing elevated eliminate emergence emission
emulate enacting enacts encouragement endure
enhance ensued entitlement entitlements envelops
equaling equilibrium erase erased erases
erode eroded erodes eroding eru
escalations euphoria evaporate exacted exceed
exceeding exceeds excitement expand expanding
expansion expansions explicit extend extends
fade fades fading fall fallhtml
falling falloff falls fallshtml falter
faltered faltering falters faltershtml fear
febrile feeble fell fillip finish
fiscal fitful fizzle fizzling flagging
flailing fluctuate fluctuates flurry forestalling
formulating fortify fragile fragilities fragility
fragmentation fragmented franchtml freefall freighthtml
frenzy frontal froth frustratingly fueling
fumble fundamentals gain gained gaining
gains gap gdp gdphtml generally

generate get go goldilocks grab
gradual gradually greenback grow growing
grows growth grudging gyrate gyrations
hamper hampering headwind headwinds heals
healthy heightens hiatus hiccup hikes
hinder hindering hindrance hitless hits
hitting hollowing huge humility hurt
hurting hydrofluorocarbon hysteria ichimoku igniting
illiquidity imbalances impediment imperiling impetus
imploded implosion import importance improve
improvement improving impulse incipient increase
increased increases increasing infinfy inflates
inflation inklings inr instability intensification
intensifies intensifying ipmfg jolt jolting
jump jumped jumping jumps keying
knock lack lackluster languish laps
laughs leadhtml leaguehtml leapfrog lessen
letdown lethargic lift limiting lingering
loonie lost lull lurching mainstay
maintain makeover marginally markedly market
mask matching materialization measurable meet
melt midst mindlessly mire miss
missed misses mistiming moderate moderated
moderately moderating moderation modernize modest
modestly momentary momentum mongering morphed
morphing motherwell motivation mounting multiply
mushroom mushroomed muted mutualization mutualized
narrow narrowing nationalistic neckline needing
negativity negligible nervousness noises nonagricultural
nonchalance nondurable nondurables nonfood nonfuel
nosedive noticeable noticeably numeric numerical
olympiacos onslaught opportunity outpace outpaces
outperformance outstrip outstrips outturn outweighing
overbought overshadowing overshadows overshoot overshooting
overshot oversized overstates overtake overtakes
overtaking overtook overvaluation overwhelms pace
panic paralleling parameter pare pared
pares patchy pattern penciling perceptible
peripheries persist persistence petered petering
phenomenon picture plausibility pleasant plummet
plummeted plummeting plunge plunged plunges
plunging positive pouch precarious precipitating
pressurehtml propel prospectshtml pull pullback
pulling pulls pummeling quicken quickened
quickening quickens raft rainfalls rainforests
raising rallied rallies rallieshtml rally

rallyhtml rallying rangebound rapid rat
rationality ravages reaccelerate reacceleration reach
react rebalancing rebasing rebound reboundhtml
rebounding recalibrate recede recession recessions
reconfirms recoveries recovery reduce reducing
reduction reductions reigniting reinforce reinvigorate
relapse relatively reminiscent rename reprieve
resilience resiliency resilient respectable respite
restrain restrained restrains restrict resulting
resurgence resurgent retarding retest retesting
retracement retreat retreated retreating retreats
retreatshtml revalue reverberate reverses revitalize
revival revive revives ricocheted rippled
ripples rise rises rising risks
robust robustness rocketing room rose
rout routhtml routs ruinous runaway
safeguard sagging sank sapped sapping
saps scorches see seemingly seeped
seesawing selloff selloffs sentiments sequester
sequesters sequestration severe shackle shake
shaky shave shock shocks shortage
shortfall shortfalls shrink shrinkage shrinking
shrinks shuddered shutout sideways significant
sink sinks skid skids skyrocket
skyrocketed skyrocketing slash slew slid
slide slides sliding slight slightly
slip slippage slipped slipping slips
slipshtml slow slowdown slowdowns slowed
slower slowhtml slowing slows slowshtml
sluggish sluggishness slump slumped slumphtml
slumping slumps snap snapped snapping
snaps snowballed snuffed snuffing soar
soared soaring soars softened softening
softness solace solid solidity somewhat
sparking spasm spate spawned spending
spike spikes spiking spillovers spiraled
spiraling spiralling spite splintering spluttering
spores spur spurred spurring spurt
sputtering sputters stabilization stabilize stabilized
stabilizes stabilizing stable stagnant stagnate
stagnated stagnates stagnating stagnation stalling
stalls stallshtml sterner stimulate stimulating
stoked stoking strains strangle streak
streakhtml streaks strength strengthen strengthened
strengthening strengthens strengthenshtml stresses strong
stronger stumble stunning stunting stuttering

stutters	subdued	subpar	subsector	subsectors
subside	subsidies	subsidy	substantial	sudden
supercommittee	supplant	supplementary	surge	surged
surges	surging	surpass	surpassed	surpassing
surplus	surpluses	surprise	sustain	sustainable
sustained	swell	swelling	swells	swingeing
swoon	swooned	swooning	sympathy	tailspin
tailwind	take	tanked	target	tariff
tariffs	temper	tempered	tempering	temporary
temptation	tenuous	tepid	thawing	threaten
threatening	thresholds	throes	throw	tightness
timidity	tmnochng	topped	topping	tops
touching	trail	trailing	trajectories	trajectory
transformation	translate	trend	trendline	trendlines
trickled	trims	tucpiy	tumble	tumbled
tumblehtml	tumbles	tumbleshtml	tumbling	tumult
turbulence	turmoil	turn	turnaround	unaffordable
unanticipated	uncertain	uncertainties	uncertainty	uncontrollable
undermine	undermining	underpin	underpricing	undershoot
undershooting	undertones	uneven	unevenness	unleashing
unsettles	unsettling	unspectacular	unstable	unsteady
upheaval	upheavals	upswing	uptick	upticks
uptrend	upturn	upward	usbodefn	utd
valuationshtml	vanish	varied	various	vat
versus	victoryhtml	vitality	volatilities	vols
vulnerabilities	vulnerability	vulnerable	waiver	wake
wane	waned	wanes	waneshtml	waning
warning	wastage	wasteful	wave	weak
weaken	weakened	weakenhtml	weakening	weakens
weakenshtml	weaker	weakness	weathering	wetness
whipsaw	whipsawing	wholesaling	widen	widening
widens	winhtml	winning	winshtml	wither
withstanding	withstands	wobble	wobbles	wobbling
worries	worsen	worsened	worsening	worsens

Appendix B

Category Tag Features

B.1 Category: new-product

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

B.2 Category: acquisition

accumulate	acquire	acquired	acquires	acquiring
acquisition	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	peaking	peaks	rally	rallying
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	decelerated	decelerating	deceleration	decline
declines	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
slowed	slowing	sluggish	slump	slumped
slumping	slumps	softer	spike	stabilizing
stagnation	subdued	triple	tumble	tumbled
tumbles	tumbling	unchanged	undermines	weak
weakening	weaker	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	claiming	claims	complaint	complaints
condatis	contended	copyright	cordance	counterclaim
counterclaims	countersued	countersuit	criminal	declaratory
defamation	defendant	defrauded	dismissed	document
documents	earthgrains	ecuadoreans	explanatory	fbi
filed	filing	fuhu	guilty	hoeffner
idna	indictment	indictments	infringement	infringements
infringers	infringing	innovatio	invalidation	ivi
jury	kiobel	kolon	lawsuit	lawsuits
leadscope	libel	listing	lodged	macrosolve
mattel	melendres	meritless	mga	ntp
nuvasive	obscenity	patent	petition	plaintiffs
presentation	proceedings	prosecutors	prospectus	racketeering

rambus	retaliated	righthaven	schelsky	sherley
slander	spansion	streeteasy	sued	suit
suits	summonses	superseding	unsealed	unsigned

B.6 Category: fiscal-report

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

B.7 Category: investment

accumulated	acquire	acquired	acquires	acquiring
add	allocate	allocated	allocates	allocating
allocation	allocations	allotted	amass	amassed
amassing	assets	assist	attract	attracted
attracting	birr	borrow	borrowed	bought
budgeted	build	buy	buys	cater
combine	complement	concentrated	consume	create
deploy	develop	devote	disbursed	dispose
distribute	divest	earmarked	earn	earned
encourage	entice	facilities	financings	funds
generate	handle	holds	install	interested

invest	invested	investing	investment	investments
invests	leftover	loaned	lure	luring
managed	manages	merge	netted	offer
offload	operate	oversees	owns	parters
portfolio	portfolios	poured	prospered	purchase
purchased	racked	raise	recycle	redeem
refurbish	repay	save	sell	sells
sold	specializes	spend	spends	spent
stakes	swapped	tap	unload	utilize

B.8 Category: bankrupt

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

B.9 Category: government

abhisit	admin	agcy	agency	austerityhtml
bailouthtml	ballot	barack	biacora	boiko
borissov	bouh	bush	cash	clinton
congress	deductions	dejan	directorate	discord
donilon	electable	election	elections	elshad
geithner	geospatial	goverment	government	governor

grishanin	gwede	hillary	ideological	ilmars
inst	jugnauth	kissinger	legislation	leon
levies	madeleine	massimov	meles	minister
minster	muhyiddin	muni	municipal	najib
nasirov	nhlanhla	nijkamp	obama	outgoing
panetta	parliamentary	participant	pjescic	plouffe
pluralism	political	pravind	president	presidential
reagan	referendum	republicans	republicanshtml	rodham
roosevelt	rousseff	rovnaq	runoff	secretary
secular	shinawatra	shinseki	shouguo	stepshtml
suthep	tax	taxes	tempore	termpittayapaisith
thaksin	thaugsuban	tpa	treasury	undersecretary
valcke	vat	vice	vote	votes
wafa	watchdog	yingluck	yoobamrung	zeljko

B.10 Category: analyst-highlights

acknowledge	acknowledged	address	addressing	affect
alienate	allaying	amid	articulate	clarify
confront	contradict	define	defuse	demonstrate
demonstrates	depict	describe	desires	determinants
diminish	discuss	downplayed	echoed	embrace
embraces	emphasize	emphasized	enhance	ensure
exaggerate	explain	exploit	expose	fueling
grasp	heightening	highlight	highlighted	however
humanize	ignore	illuminate	illustrate	illustrates
imply	instigate	intensified	interpret	maintain
marginalize	militarily	motivate	nonetheless	object
observe	offend	overcome	overestimate	possess
prove	provoke	recognize	reflect	reflects
reignited	reigniting	reinforce	reinforced	reinforces
reiterate	relate	resist	reveal	shortcomings
showing	signal	solve	sparked	stoking
strengthen	stressing	suggest	suggested	suggesting
suggests	tackle	tempering	underestimate	underline
underlined	underlines	underlining	undermine	undermined
undermines	underscore	underscored	underscores	underscoring