Stock Price Prediction via Deep Belief Networks

by

Xi Chen

Master of Econometrics [Wuhan University of Technology, 2012]

A Report Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Business Administration

in the Faculty of Business Administration

Supervisor: Donglei, Du, PhD, Faculty of Business Administration

Examining Board: Jeffery McNally, PhD, Faculty of Business Administration, Chair
Eben Otuteye, PhD, Faculty of Business Administration
Huajie Zhang, PhD, Faculty of Computer Science

This report is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2015

ABSTRACT

Artificial Intelligence (AI) techniques such as Neural Network (NN) have been widely used in the financial industry to predict stock prices to aid investment decisions. However, the traditional NN has been quickly surpassed by the new rapid developing Deep Belief Network (DBN) and its variants in terms of prediction accuracy in areas like image processing and speech recognition. This project aims to apply the DBN technique to stock price prediction and compare its performance with the traditional NN. In particular, we use the S&P500 index as a case study, and our numerical results show that DBN indeed performs better than the traditional NN. Hence, as a new generation of AI technique, DBN shows great promise in stock price prediction and forecasting.

Keywords: Finance, Neural networks, Deep belief networks, Technical analysis, Prediction

ACKNOWLEDGEMENTS

My deepest gratitude goes first and foremost to Professor Donglei Du, my supervisor, for his constant encouragement and guidance. He has taught me the methodology to do the project, without his consistent and illuminating instruction, this report could not have reached its present form. Also I would like thank Dr. Eben Otuteye and Dr. Jeffery McNally for serving on the examination board.

Secondly, I would like to express my great gratitude to Yong Jin, a PhD student in the research area of Data Mining at Computer Science of University of New Brunswick who helped me on how to proceed with the technical implementations.

Lastly my thanks would go to Marilyn Davis, the MBA program assistant at Faculty of Business Administration, for her kindhearted instruction of the related information and timeline of the project.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Goals of the project

Artificial Intelligence (AI) techniques such as Neural Networks (NN) have been widely employed in the financial industry to predict stock prices to aid investment decisions. However, the traditional NN has been surpassed by the new rapid developing Deep Belief Network (DBN) and its variants in terms of prediction accuracy in areas like image processing and speech recognition.

This project aims to examine the use of DBN in stock price prediction and compares its performance with traditional NN. In particular, we will use S&P500 index as a case study to illustrate that DBN can indeed provide more accurate predication compared to the traditional NN technique.

1.2 Background and literature review

Financial prediction plays a critical role in investment management. Typically, there are three types of paradigms for financial market analysis, namely Fundamental Analysis (FA), Technical Analysis (TA), and Quantitative Analysis (QA). FA refers to the analysis of a business's financial statements such as balance sheets, income statements, equity statements, ratios and etc.. TA is a methodology of investigating historical price and /or volume movements of financial markets attempting to predict future price trend. QA applies scientific methods, such as statistics and applied mathematics, to help make investment decisions. The methods of NN and DBN discussed in this report belong to the category of intersection of both TA and QA. In particular, both NN and DBA are AI algorithms that use price indicators for price prediction.

There are many extant works on applying NN technique to stock price prediction. Joseph (1996) proposes different neural network techniques for financial prediction including the trend prediction and price prediction. The inputs are mainly generated from price data. Edward (1996) describes the methodology to develop neural network to accurately predict the financial market. It uses commercially available software called NeuroShell to build, train and test the neural network model. Its inputs are not only generated from the prices of one stock / index, but also from some other macroeconomic indicators. Chun (2004) applies machine learning / data mining techniques to discover the effect of a number of possible scenarios including the selection of learning approaches, the choice between NN or rule-based reasoning technique, the adaptation of active or passive trading strategy, etc. On the other hand, Bengio (2007) demonstrates that DBN with unsupervised training can perform better than ordinary NN with only back propagation training approach. Hence DBN is performed to analyze financial market in this study compared with NN.

1.3 Organization of the report

The rest of this report contains the following sections. In Section 2, we review the methodologies of NN and DBN. In Section 3, we describe the implementation of these two methods applied to stock price prediction. In Section 4, we perform the case study by applying and comparing the predication accuracy of NN and DBN to the S&P500 index. Finally in Section 5, we conclude this project by offering directions of future work.

2. Theoretical frameworks

In this section, we will describe the two network models of NN and DBN, including the structures and learning algorithms.

2.1 Neural networks

The NN was originally innovated to imitate the work of human brain, which can express complex connections among variables including linear or nonlinear relationships. The NN consider the nodes as ‘Artificial Neurons’, hence it can also be called artificial neuron networks (ANNs). It is constructed to imitate human brain's interconnected system of neurons. Specifically, an artificial neuron is a mathematical model inspired from human natural neuron in Fig 2.1.

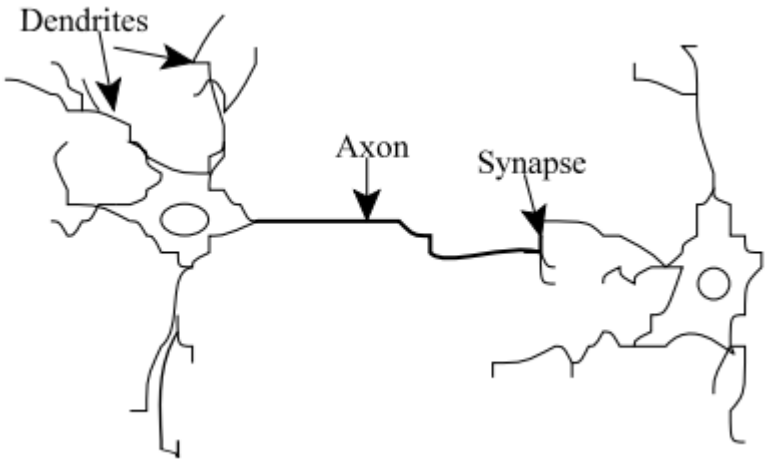

Figure 2.1 Natural Neurons (Source: Gershenson, 2003)

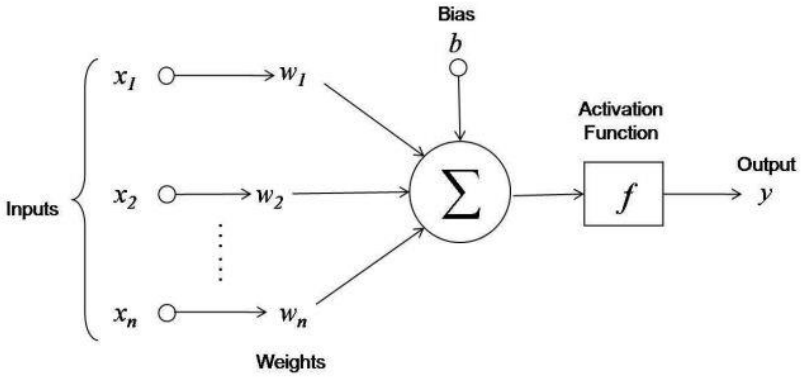

Figure 2.2 Artificial neuron model

When an artificial neuron is introduced into modeling, the real neuron's complexity is highly abstracted. In detail, as in Fig 2.2, the *inputs* are abstracted from synapses, which will be multiplied by their corresponding *weights*. Then they are calculated through a mathematical function, usually a linear combination of the inputs including an offset term, also there is another function called *activation function* which determines the activation value of the neuron, which is actually the *output* corresponding to the inputs. Through this artificial neuron, each output can be expressed to have a non-linearity relationship with the combination of all the inputs. Mathematically, the neuron model can be generalized in Eq.(1) and Eq.(2).

$$\sum = b + x_1 \cdot w_1 + x_2 \cdot w_2 + ... + x_n \cdot w_n \tag{1}$$

$$y = f(\sum) \tag{2}$$

Normally the activation function in NN is represented by sigmoid function (or logistic function) $f(x) = 1 / (1 + \exp(-x))$, which is plotted in Fig 2.3. In this graph, the artificial neuron is mathematically activated when the output y is greater than 0.5, otherwise it will be inactivated.

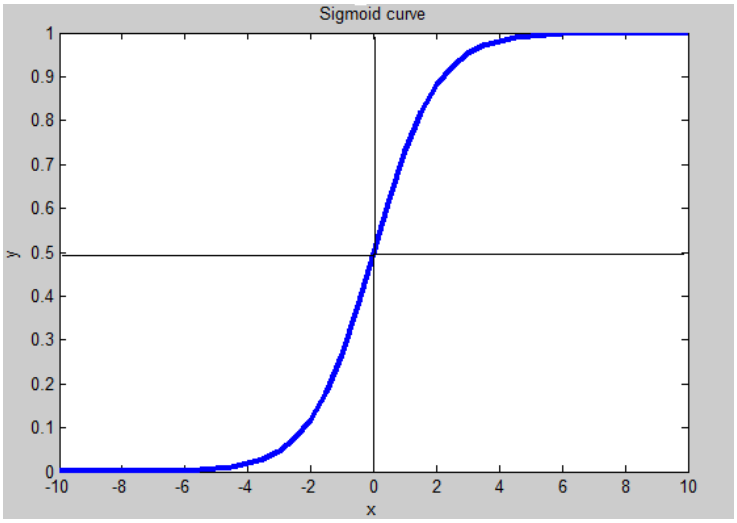

Figure 2.3 Curve of sigmoid function

Therefore, through the introduction of artificial neuron and understanding of its work mechanism, I will describe a basic neural network structure including one input layer, one hidden layer and one output layer as depicted in Fig 2.4. It is actually a directed graph from bottom to up, where each arrow indicates a directed connection between the output of one neuron in current layer and the input of another neuron in next layer.

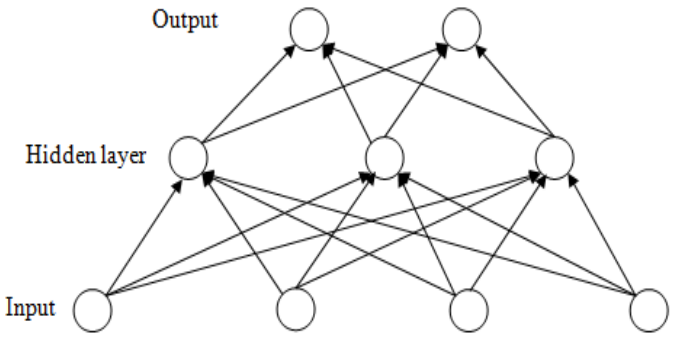

Figure 2.4 Neural Networks

Next we will discuss how to train the NN. This project will use Back propagation algorithm to train NN which is introduced in UFLDL tutorial[1]. The following part is extracted from this tutorial or inspired by it.

Let N training examples be represented by $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$, we define the cost function for a single training example (x, y) by Eq.(3).

$$J(W; x, y) = \frac{1}{2} \| f_W(x) - y \|^2 \tag{3}$$

Where W is the weight matrix including the bias term. It is one-half squared error cost function. Then the overall cost for all the training examples is given by Eq.(4).

$$J(W) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{2} \| f_W(x^{(i)}) - y^{(i)} \|^2 \right) \tag{4}$$

Then our goal is to minimize $J(W)$ as a function of W. To train the neural network, we should initialize the parameter W within a small random range around zero. Considering $J(W)$ is a non-convex function, gradient descent is applicable here.

One epoch (also can be called as iteration) of gradient descent updates the weights following the Eq.(5).

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W) \tag{5}$$

Where α is the learning parameter. Then we only need to find the partial derivative of $J(W)$with respect to W at layer l. This is the goal of back propagation. Firstly, we will perform feed forward pass based on the initialized weights. Then we need to compute the

[1] The UFLDL tutorial website: http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm.

error between the network output and real output. In detail, suppose the output for the node j is represented by f_j after using the activation, while the desired output of node j is Oj, then the error term at node j is given by Eq. (6). To avoid mathematical expression complexity, the layer label l is omitted in later formulas.

$$\delta_j = \frac{\partial J(W)}{\partial O_j} = f_j - O_j \tag{6}$$

Note, when referring to the output layer, then the desired output of node j is actually y_j. The partial derivative of $J(W)$ with respect to W can be divided into two derivatives' multiplication in Eq. (7).

$$\frac{\partial}{\partial W_{ij}} J(W) = \frac{\partial J(W)}{\partial O_i} \cdot \frac{\partial O_i}{\partial W_{ij}} \tag{7}$$

Similar to Eq. (6), the first factor if the error term of node i, while the second factor can be re-written separately as Eq.(8).

$$\frac{\partial O_i}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_{k \in Ai} W_{ik} a_k = a_j \tag{8}$$

Where Ai represents all the nodes j anterior to node i in the next layer, and a_k refers to the activation value of node k.

Then the partial derivative of $J(W)$ regarding W can be calculated through Eq.(9).

$$\frac{\partial}{\partial W_{ij}} J(W) = \delta_i a_j \tag{9}$$

After calculating the Eq.(9), the weights can be updated through Eq.(5). Specifically the error term can be first calculated from the output layer because the real output y is given in the training data, and then extended to the previous layers. The weights are updated

until the cost function converges or the cost function remains in a predefined tolerance range.

2.2 Deep belief networks

The DBN is usually referred to as deep neural networks, and so they are normally viewed as the advancement of NN. Hence they have similar structures and components like artificial neurons. One major point is that DBN has introduced Restricted Boltzmann Machine (RBM) into the system, which distinguishes it from NN. The DBN structure including one RBM is depicted in Fig 2.5. The only difference between neural network and DBN is that the RBM is a bipartite graph (see the difference between Fig 2.4 and Fig 2.5), which shows symmetric connection between input and hidden layer (or two hidden layers). Each RBM is to model the distribution of its input, using an unsupervised training method.

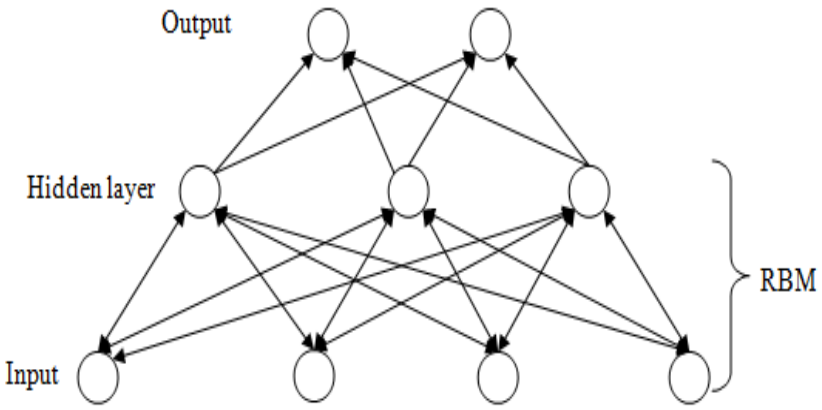

Figure 2.5 Deep Belief Networks

Considering the special structure in DBN, it is also trained different from NN. Unlike supervised back propagation, DBN is trained in a greedy layer-wise fashion (Hinton,

2006). There are basically two training phases in DBN training: RBM unsupervised training and supervised back propagation.

The first phase, RBM unsupervised training, is actually to obtain a higher abstracted representation of the input (or the input features of previous layer) without using the target characteristics. Specifically, let the input of RBM be represented by vector *x*, and the hidden layer as vector *h*, also suppose the matrix weights *W* associated with the connection between *x* and *h*, bias term for *x* and *h* is vector *a* and *b* respectively. Then the energy of the RBM configuration is written by Eq.(10).

$$E(x,h) = -a^T x - b^T h - x^T W h \tag{10}$$

Where the right corner symbol of *T* means transpose of the matrix or vector. Then the joint probability distribution over input and hidden vectors is defined as Eq.(11) in terms of the energy function.

$$P(x,h) = \frac{1}{Z} e^{-E(x,h)} \tag{11}$$

Where Z is a partition function as the sum of $e^{-E(x,h)}$ over all possible values of *x* and *h*. Then the marginal probability for the input is to sum $p(x, h)$ over all hidden units, which is actually described in Eq.(12).

$$P(x) = \sum_h P(x,h) = \frac{1}{Z} \sum_h e^{-E(x,h)} \tag{12}$$

Hence the training goal for RBM is to maximize the product of probabilities over some training set *X* with respect to its weights matrix *W* as Eq.(13).

$$\arg\max_W \prod_{x \in X} P(x) \tag{13}$$

Hinton (2002) has proposed contrastive divergence (CD) and Gibbs sampling algorithm to train RBM and also he provided a detailed practical guide to train RBM in 2010.

The second phase is supervised back propagation, which is similar to the previous part. The only difference is that, we do not need to randomly generate initial weights for NN (except the weights to output layer), but to transfer the weights obtained in RBM unsupervised training into this phase, and then back propagation is applied to fine-tune the weights parameters.

3. Methodology

This section discusses the detailed methodologies to perform NN and DBN for price prediction of S&P500 index. It introduces how to collect data and preprocess it, and also how to design the NN and DBN.

3.1 Data collection and preprocessing

The S&P500 index data is collected from Yahoo Finance through the quantmod package of R programming language. In order to obtain enough information from the price data, I collect the data from Jan. 1st, 2000 to Dec. 31st, 2014, including the variables of Date, Open, High, Low and Close. To successfully apply the historical data to predict the future market, the whole data set is split into two parts: training set and test set, the former one is from Jan. 1st, 2000 to Dec. 31st, 2011 and the latter one includes data from Jan. 1st, 2012to Dec. 31st, 2014.

Data preprocessing is to generate feasible input pattern and target pattern for NN and DBN. For the input, due to the different expressive power, it will include two types: (1) linear transformation; and (2) logarithm transformation.

(1) Linear input

This strategy is to create input pattern using the daily change in the four types of prices including open price, close price, high price, and low price, as illustrated in Table 3.1. However, they cannot be directly applied in the network since the data may occur in

different scales. Then Joseph (1996) has introduced a pre-calculated scaling factor called probable volatility window (λ) to normalize the data. For each type of Δ (open, close, high, and low) in Table 3.1, the probable volatility window is computed through Eq. (14).

$$\lambda = \frac{2}{N}\sum_{i=1}^{N}\frac{|\Delta_i|}{c_i} \tag{14}$$

Where N is the number of days being analyzed, and i represents Day i.

Table 3.1 Linear input pattern

Neuron	Pattern	Value(linear)
1	△Open	*o*(i+1) - *c*(i)
2	△Close1	*c*(i) - *c*(i-1)
3	△High1	*h*(i) - *h*(i-1)
4	△Low1	*l*(i) - *l*(i-1)
5	△Close2	*c*(i-1) - *c*(i-2)
6	△High2	*h*(i-1) - *h*(i-2)
7	△Low2	*l*(i-1) - *l*(i-2)
8	△Close3	*c*(i-2) - *c*(i-3)
9	△High3	*h*(i-2) - h(i-3)
10	△Low3	l(i-2) - l(i-3)

Then the actual input value for its corresponding type of Δ is calculated by Eq.(15).

$$x_i = \frac{\Delta_i}{c_i\lambda}, i = 1,...,N \tag{15}$$

Where x_i is the i^{th} data point value for its corresponding type of input. Please note that x_i will not be applicable for the previous two days change when i equals 1, 2, or 3, and also it will not be applicable for the last day because ΔOpen is computed through the result of

the last day's open price minus its previous day's close price, while the other nine inputs' computations do not need the last day's price. This feature is important because the opening price of the day represents the closing price from all overnight trading which has taken place.

Table 3.2 Probable volatility calculation for each type of input (part of 2014)

Date	Open	High	Low	Close	△Open	∆ Close1	△High1	∆ Low1	∆ Close2	∆ High2	∆ Low2	∆ Close3	∆ High3	∆ Low3
2014-12-10	2058.86	2058.86	2024.26	2026.14	1.78									
2014-12-11	2027.92	2055.53	2027.92	2035.33	-4.97	9.19	-3.33	3.66						
2014-12-12	2030.36	2032.25	2002.33	2002.33	2.7	-33	-23.28	-25.59	9.19	-3.33	3.66			
2014-12-15	2005.03	2018.69	1982.26	1989.63	-2.92	-12.7	-13.56	-20.07	-33	-23.28	-25.59	9.19	-3.33	3.66
2014-12-16	1986.71	2016.89	1972.56	1972.74	1.03	-16.89	-1.8	-9.7	-12.7	-13.56	-20.07	-33	-23.28	-25.59
2014-12-17	1973.77	2016.75	1973.77	2012.89	6.09	40.15	-0.14	1.21	-16.89	-1.8	-9.7	-12.7	-13.56	-20.07
2014-12-18	2018.98	2061.23	2018.98	2061.23	-0.19	48.34	44.48	45.21	40.15	-0.14	1.21	-16.89	-1.8	-9.7
2014-12-19	2061.04	2077.85	2061.03	2070.65	-1.37	9.42	16.62	42.05	48.34	44.48	45.21	40.15	-0.14	1.21
2014-12-22	2069.28	2078.76	2069.28	2078.54	2.94	7.89	0.91	8.25	9.42	16.62	42.05	48.34	44.48	45.21
2014-12-23	2081.48	2086.73	2079.77	2082.17	1.08	3.63	7.97	10.49	7.89	0.91	8.25	9.42	16.62	42.05
2014-12-24	2083.25	2087.56	2081.86	2081.88	2.42	-0.29	0.83	2.09	3.63	7.97	10.49	7.89	0.91	8.25
2014-12-26	2084.3	2092.7	2084.3	2088.77	-1.14	6.89	5.14	2.44	-0.29	0.83	2.09	3.63	7.97	10.49
2014-12-29	2087.63	2093.55	2085.75	2090.57	-2.08	1.8	0.85	1.45	6.89	5.14	2.44	-0.29	0.83	2.09
2014-12-30	2088.49	2088.49	2079.53	2080.35	1.76	-10.22	-5.06	-6.22	1.8	0.85	1.45	6.89	5.14	2.44
2014-12-31	2082.11	2085.58	2057.94	2058.9										
λ= Eq.(14)					0.0023	0.0152	0.0093	0.0134	0.0155	0.0097	0.0140	0.0167	0.0105	0.0151

Table 3.2 shows how to apply Eq.(14) to compute the probable volatility window values (λ) for each type of ∆, and after that the input pattern of X1 through X10 is computed via Eq.(15) as listed in Table 3.3.

Table 3.3 Excerpt from a price prediction linear input pattern (part of 2014)

Date	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
2014-12-10	0.3863									
2014-12-11	-1.0737	0.2975	-0.1754	0.1339						
2014-12-12	0.5929	-1.0860	-1.2465	-0.9517	0.2955	-0.1714	0.1303			
2014-12-15	-0.6453	-0.4206	-0.7307	-0.7512	-1.0678	-1.2061	-0.9165	0.2761	-0.1597	0.1216
2014-12-16	0.2296	-0.5642	-0.0978	-0.3662	-0.4145	-0.7085	-0.7250	-1.0000	-1.1258	-0.8578
2014-12-17	1.3303	1.3144	-0.0075	0.0448	-0.5402	-0.0922	-0.3434	-0.3772	-0.6426	-0.6593
2014-12-18	-0.0405	1.5454	2.3135	1.6333	1.2541	-0.0070	0.0418	-0.4899	-0.0833	-0.3112
2014-12-19	-0.2909	0.2998	0.8605	1.5122	1.5030	2.2143	1.5559	1.1592	-0.0064	0.0386
2014-12-22	0.6219	0.2501	0.0469	0.2956	0.2918	0.8242	1.4416	1.3904	2.0414	1.4383
2014-12-23	0.2281	0.1149	0.4104	0.3752	0.2440	0.0451	0.2823	0.2705	0.7615	1.3354
2014-12-24	0.5111	-0.0092	0.0427	0.0748	0.1123	0.3946	0.3591	0.2266	0.0417	0.2620
2014-12-26	-0.2400	0.2174	0.2638	0.0870	-0.0089	0.0410	0.0713	0.1039	0.3640	0.3321
2014-12-29	-0.4375	0.0567	0.0436	0.0516	0.2122	0.2534	0.0832	-0.0083	0.0379	0.0661
2014-12-30	0.3720	-0.3237	-0.2608	-0.2226	0.0557	0.0421	0.0497	0.1980	0.2357	0.0776
2014-12-31										

(2) Log input

Logarithms can be used to extract features in financial prediction. Joseph (1996) has proved the logarithm can be applied to compress the input values and may have a greater impact on the learning algorithm. Hence the basic linear approach is modified to use the ratios and then calculate the input based upon the logarithm transformations of ratios. The log input is depicted in Table 3.4.

Table 3.4 Logarithm input pattern

Neuron	Pattern	Value(log)
1	△Open	*o* (i+1) / *c* (i)
2	△Close1	*c* (i) / *c* (i-1)
3	△High1	*h* (i) / *h* (i-1)
4	△Low1	*l* (i) / *l* (i-1)
5	△Close2	*c* (i-1) / *c* (i-2)
6	△High2	*h* (i-1) / *h* (i-2)
7	△Low2	*l* (i-1) / *l* (i-2)
8	△Close3	*c* (i-2) / *c* (i-3)
9	△High3	*h* (i-2) / *h* (i-3)
10	△Low3	*l* (i-2) / *l* (i-3)

For the target output, it is generated by grouping the price percent change into some predefined intervals. Corresponding to the input, the price percent change of each data point is calculated using $[c(i+1)-c(i)]/c(i)$. Based on the book of Joseph (1996), the predefined price percent change intervals were originally grouped into four categories, but considering the complexity of multi-class classification and the necessity of reality, the price change intervals are manually defined into four intervals that are illustrated in Table 3.5. Specifically, there are four columns to represent the four predefined ranges, and each column represents one class label. For the first column of each data point, when the class label is -2, its target value is 1, or it should be 0; for the second column of each data point, when the class label is -1, its target value is 1 while others should 0. The left two columns are in a similar fashion. Some samples output pattern is depicted in Table 3.6 as well.

Table 3.5 Price percent change predefined intervals

Range	<-0.5%	-0.5% to 0	0 to 0.5%	>0.5%
Class label	-2	-1	1	2
Meaning	Strong decrease	Minor decrease	Minor increase	Strong increase

Table 3.6 Excerpt from a price prediction output pattern (part of 2014)

Date	Percent change	Range label	-2	-1	1	2
2014-12-10	0.454%	1	0	0	1	0
2014-12-11	-1.621%	-2	1	0	0	0
2014-12-12	-0.634%	-2	1	0	0	0
2014-12-15	-0.849%	-2	1	0	0	0
2014-12-16	2.035%	2	0	0	0	1
2014-12-17	2.402%	2	0	0	0	1
2014-12-18	0.457%	1	0	0	1	0
2014-12-19	0.381%	1	0	0	1	0
2014-12-22	0.175%	1	0	0	1	0
2014-12-23	-0.014%	-1	0	1	0	0
2014-12-24	0.331%	1	0	0	1	0
2014-12-26	0.086%	1	0	0	1	0
2014-12-29	-0.489%	-1	0	1	0	0
2014-12-30	-1.031%	-2	1	0	0	0
2014-12-31						

3.2 Network configuration

After data collection and preprocessing work, this phase is to configure the NN and DBN. Both NN and DBN need to determine the following characteristics:

- The number of input neurons;
- The number of output neurons;
- The number of hidden layers;
- The number of hidden neurons in each hidden layer;

The number of input neurons for price prediction network is defined based on the input pattern selected in Table 3.1 and Table 3.4 respectively. Those 10 inputs features capture the preceding three days of price movement. The number of output units should be the number of price percent change ranges, which is 4. For the number of hidden layers, considering there is only 10 inputs patterns and 4 output patterns, then to avoid over-

fitting in the network training, only one hidden layer is sufficient for the price prediction network. Finally, regarding that some correlations exist among the 10 input patterns since the three-day price moving change is used to compute the inputs, hence the number of hidden neurons should be less than or equal to 10. In my implementations, I have tried numbers of hidden neurons through 4 to 10, and then choose the best number, which is 8. In summary, for the purpose of fair comparison, both NN and DBN will have the same structure in our system as described in Fig 3.1. The difference is that the traditional NN is unidirectional from input to output, but DBN is bidirectional between input and hidden layer (one RBM), and hidden layer to output layer (four ranges) is unidirectional.

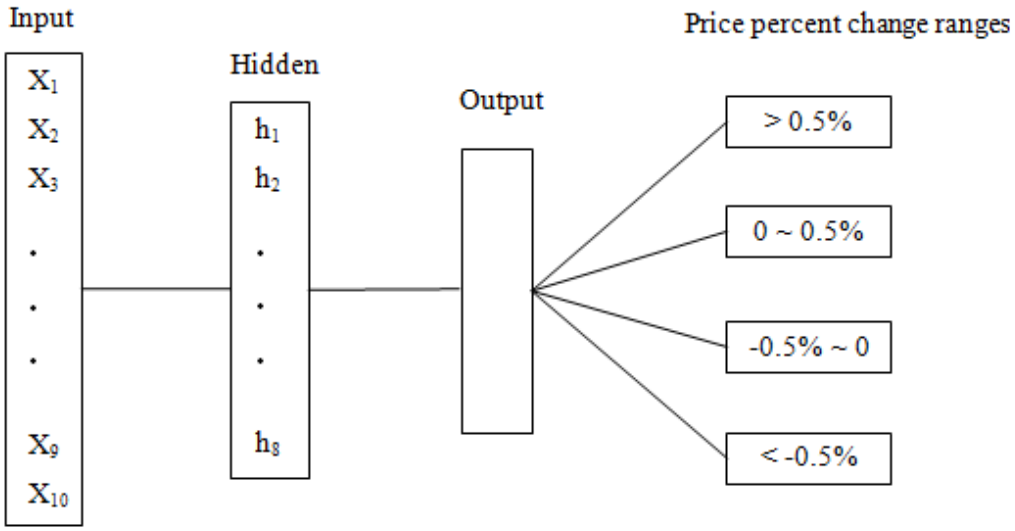

Figure 3.1 Network structure configuration

4. Experimental results and analysis

This part describes the experiments discussed in section 3. There are two types of input patterns, and each type of input is performed around ten times using NN and DBN respectively. For neural network settings, the learning rate is set to 0.5 and momentum to 0.05. For RBM part in DBN training, we set learning rate to 0.5 and momentum to 0.01. Based on the MATLAB R2014a software and Intel (R) Core i5-3337U CPU (Central Processing Unit), 12.0GB RAM (Random-Access Memory) and 64-bit Windows Operating system, the training summary is depicted in Table 4.1.

Table 4.1 Training summary

	NN-linear	DBN-linear	NN-log	DBN-log
Training set	3016 (2000-2011)			
Test set	753 (2012-2014)			
Learning epochs	1500	10 + 1200	3500	10 + 3000
Average training time	6.0 Mins	5.3 Mins	13.1 Mins	11.5 Mins
Average error rate	47.3%	37.8%	48.7%	42.4%

In this table, NN-linear represents the linear input for NN, and the other three columns have similar meanings. The training set is split into 3016 examples and 753 examples for test set. For the linear input, the NN model needs to run 1500 epochs until the model converges around 6 minutes, and its error rate is 47.3%; however, the DBN has run 10 epochs for RBM unsupervised training and 1200 epochs for supervised back propagation training to reach a better error rate of 37.8% within a little less training time at 5.3 minutes. For the log input, the ordinary NN model needs to run 13 minutes through 3500 epochs until convergence, reaching an error rate of 48.7%, but the DBN can obtain 42.4% error rate, which is better than NN. On the other hand, the linear input cost less training time than the log input but it has a relatively better result.

Furthermore, the prediction accuracy on each price percent change range is illustrated in Table 4.2. And for linear input pattern and logarithm input pattern we have also drawn a comparison graph for each range accuracy in Fig 4.1 and Fig 4.2. They show the DBN performs better than NN for each input.

Table 4.2 Price prediction accuracy on each range

Range	NN-linear	DBN-linear	NN-log	DBN-log
<-0.5%	0.485	0.659	0.394	0.561
-0.5% to 0	0.477	0.457	0.422	0.663
0 to 0.5%	0.602	0.619	0.428	0.352
>0.5%	0.516	0.774	0.801	0.780
Overall average	0.527	0.622	0.513	0.576

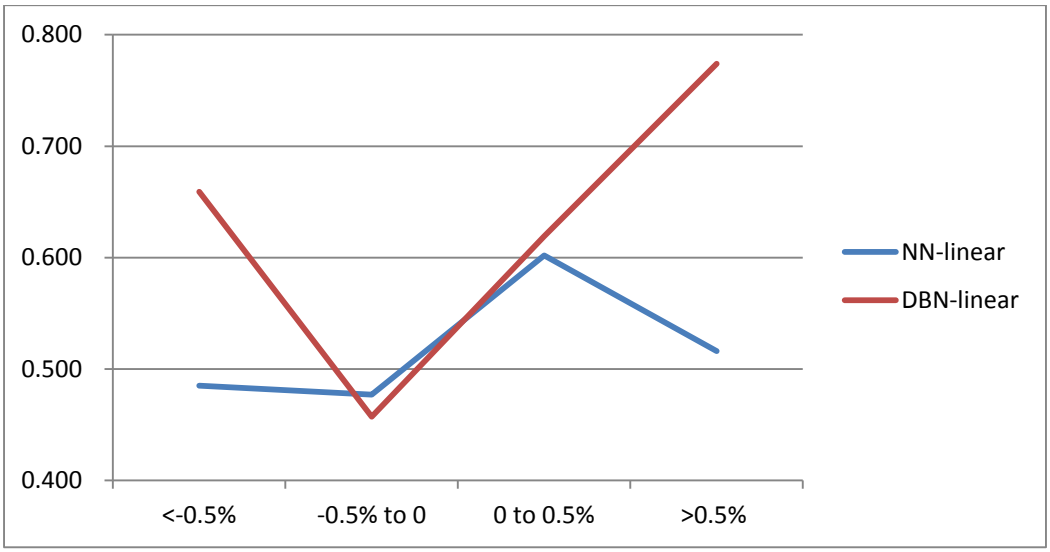

Figure 4.1 Accuracy comparison for linear input

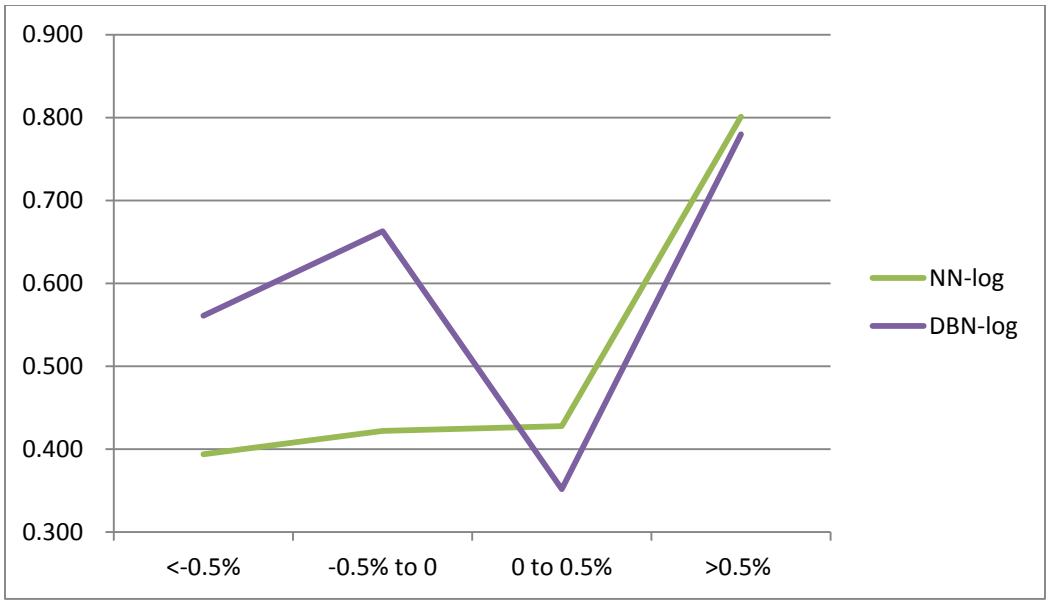

Figure 4.2 Accuracy comparison for log input

Finally, as depicted above, the linear input performs better in price prediction. It indicates the price differences between two days have more impact on the price than the price ratios between two days. In other words, the trend of the price in a short period (three days here) provides a greater prediction result on the following day trading approach. When next day trading method is to be made, the linear input should be calculated for further decision making. For example, when we use linear input applied in DBN, a data point has a prediction accuracy of 77.4% in the range that is greater than 0.5%, which will lead us to make a relatively reasonable trading approach.

5. Conclusions and future work

5.1 Conclusions

This study has shown the applications of NN and DBN in stock price prediction. Compared to NN, it is clear that DBN can better forecast the financial price within a predefined price percent change range if the raw data is properly preprocessed and the model is properly trained. Overall, based on the experiments in this study, we can reach the following conclusions:

- DBN performs better than NN in price prediction because the RBM unsupervised training can help offer better initial weights for the supervised back propagation training;
- The linear input pattern has a better impact on the prediction results compared to the logarithm input pattern;
- The DBN using the linear input pattern can give a better result for our investment strategy.

To conclude, the DBN reaches a better accuracy rate of 62.2%, which means we have 62.2 percent probability to correctly predict the price percent change at a predefined range. More specifically, if we collect data in a proper period and process it using the linear approach, we can predict it in a range with a certain probability. For example, given the prices (high, low, and close) of Day *i*, *i*-1, *i*-2, and *i*-3, and also the open price of Day *i*+1, we can use this system to predict the price change for the close price of Day *i*+1 as soon as the open price is known to us. However, we should collect the right raw data and repeat training until a good model parameter set is obtained. Also the best test

method of this forecasting tool is to investigate during practical investment which may take some days, months or even years.

5.2 Future work

This work has provided a comparison between NN and DBN applied for price prediction in financial market. There are still much room for improvement. For the input pattern, our input pattern only covers three day price changes due to the time restriction of this study. Practically, more days like five days or ten days can be tried in future, but a long period price trend (e.g. six months period price moving average) is not feasible because it will easily result in over-fitting issues. Also the exponential or Fibonacci techniques have also proved be effective because they can integrate the trend information into the system. On the other hand, before we design NN or DBN for financial price forecasting, we need to have a comprehensive consideration.

In summary, if we want to apply this financial price prediction tool into our investment strategy system, we should consider the following characteristics:

- More days' price trend should be considered for input pattern, like ten days' price change, or exponential price change, which will generate more input features;
- Get more familiar with DBN and make sure the input pattern is feasible for this system;
- Carefully design the network structure, including activation functions, the number of hidden layers and the number of hidden units in each layer. Also the layout network structure relates to the real situation in financial market, such as trading volume, share price, P/E ratios, industry analysis and etc.;
- Once a network is developed, we should perform many runs to verify the results

and compare them among different input patterns;

- This prediction tool should be updated when we collect more recent data because recent data always has greater impact on the prediction.

This report is from a technical scope to analyze financial market. The higher accuracy of DBN gives us a relative better trend for stock price prediction. Nevertheless, as known to all, the investment strategy is never easy to decide because actual trading always keeps changing. In future, the combination of price changes and other kinds of input, and carefully designed network structure may lead us to a better prediction result.

Bibliography

Zirilli, J. S. (1996). *Financial prediction using neural networks*. London: International Thomson Computer Press.

Gately, E. (1996). *Neural networks for financial forecasting*. New York: Wiley.

Chun, S.-H., & Kim, S. H. (February 01, 2004). Data mining for financial prediction and trading: application to single and multiple markets. *Expert Systems with Applications, 26*,2, 131-139.

Bengio, Y., Lamblin, P., Popovici, D., &Larochelle, H. (January 01, 2007). Greedy Layer-Wise Training of Deep Networks. *Advances in Neural Information Processing Systems*, 19, 153-160.

Gershenson, C. (2003). *Artificial neural networks for beginners*. arXiv preprint cs/0308031.

Kutsurelis, J. E. (1998). *Forecasting financial markets using neural networks: An analysis of methods and accuracy* (Doctoral dissertation, Monterey, California. Naval Postgraduate School).

Hinton, G. E., Osindero, S., & Teh, Y. W. (January 01, 2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 7, 1527-54.

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. *Momentum*, *9*(1), 926.

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, *14*(8), 1771-1800.

Curriculum Vitae

Candidate's full name: Xi Chen

Universities attended:

Dec. 2011, Master in Econometrics, Wuhan University of Technology

May 2009, Bachelor in Finance, Huazhong University of Technology, Wenhua College