A STUDY OF NEURAL NETWORK APPLICATION IN BEATING STOCK INDICES

KOU HAN

NATIONAL UNIVERSITY OF SINGAPORE

2017

A STUDY OF NEURAL NETWORK APPLICATION IN BEATING STOCK INDICES

Kou Han

(*M.Sc., NUS*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MATHEMATICS

NATIONAL UNIVERSITY OF SINGAPORE

2017

Supervisor:

Assistant Professor Zhou Chao

Examiners:

Dr Tong Xin

Dr Thiery, Alexandre Hoang

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Kou Han

5th December 2017

Acknowledgment

I would like to thank my supervisor, Assistant Professor ZHOU Chao for his guidance and encouragement.

I would like to thank all my department-mates and my friends in Singapore for their friendship and so much kind help.

Lastly, I would like to thank my parents for their continuous support.

Contents

Abstract

Neural networks have been applied to financial applications more and more recently, such as pricing securities, risk management and constructing portfolios. Compared with standard financial methods, neural network can take all the data relevant into account and it can learn not only linear relationships but more complex features of the input data. Another advantage of neural network is that it is more easy to reduce over-fitting and improve the performance on the test set. In this study, we focus on the application of neural networks in constructing portfolios of stocks which can outperform the stock index by a specified level. The results show that the deep portfolios do outperform the Hang Seng index and Shanghai Stock Exchange 180 index. The study also shows that we can make profits by longing the portfolio and shorting the exchanged traded fund (ETF) of index.

List of Figures

List of Symbols

X	input
Y	output
$f^{(l)}$	activation function
$W^{(l)}$	weight
$b^{(l)}$	bias
$a^{(l)}$	activation value

List of Abbreviations

BGD	Batch Gradient Descent
SGD	Stochastic Gradient Descent
MBGD	Mini-Batch Gradient Descent
HSI	Hang Seng index
DPT	Deep Portfolio Theory
SSE	Shanghai Stock Exchange
ETF	Exchanged Traded Fund

Chapter 1

Introduction

Neural networks have been explored to be applied in financial applications more and more recently, such as pricing securities, risk management and constructing portfolios. Financial problems always involve large datasets and complex data interactions. Compared with standard methods in finance, neural network can take all the data relevant into account and it can learn not only linear relationships but more complex features of the input data. Another advantage of neural network is that it is more easy to reduce over-fitting and improve the performance on the test set. In this study, we focus on the application of neural networks in constructing portfolios of stocks.

Modern portfolio theory was raised by Markowitz in 1952 [1]. The main idea of the theory is to reduce the risk by diversifying financial assets. However, the Markowitz model may cause large error in practice since it is sensitive to the input. Black and Litterman (1991) [2] developed the Black-Litterman model based on Markowitz's modern portfolio theory. They took the investors' views into account.

Traditional financial models, including the Markowitz model and Black-

Litterman model, are always of shallow architectures, with at most two layers. Heaton, Polson and White (2016) [3] give us a new viewpoint of constructing portfolios which is called deep portfolio theory (DPT). The 'deep' here refers to the deep architecture of the neural networks. The key idea of DPT is to get a more information-effective representation of the market by auto-encoder which is a neural network of a special bottleneck structure. The advantage of using a neural network compared with the traditional financial models is that overfitting is more easily reduced by adding a regularization term or by adopting the technique dropout. Then the DPT can be used to reach a given goal, for instance, to outperform a given stock index by a specified level.

The rest of the thesis is organized as follows. In Chapter 2 we introduce the neural networks which will be used in this study, as introduced by Heaton, Polson and White (2016) [4]. Specifically, we will introduce the feed-forward neural network and auto-encoder. We will also introduce some techniques used in training the architecture of neural networks, such as k-fold cross validation and dropout. In Chapter 3 we introduce the Markowitz modern portfolio theory and Black-Litterman model. Then we explain the DPT and the relationship between DPT and the traditional models. In Chapter 4, we show that the portfolios we constructed can beat the given stock indices. Then we can make profits by longing the portfolio and shorting the exchanged traded fund (ETF) of the given index. Finally, some conclusions are presented in Chapter 5.

Chapter 2

Neural Networks

2.1 General structures of neural networks

Neural network, also called artificial neural network, is a form of machine learning. Machine learning is training models with the input data and then making predictions from new data based on the trained models. The main problem of machine learning as well as neural network is to find an input-output mapping

$$Y = F(X)$$

where X is the input and Y is the output. The input X can be high-dimensional, and the output Y can be continuous or discrete. When Y is discrete, this corresponds to classification problem. In this study, the outputs are prices of stocks and stock indices, thus are continuous.

In this study, we used supervised feed-forward neural networks. For supervised algorithm, the actual targets Y are given in training datasets, and we want to find a predictor denoted by $\hat{Y} := F(X)$ to minimize the distance between Y and $\hat{Y}$.

The general structure of the neural network in this study can be shown

in Fig. 2.1.

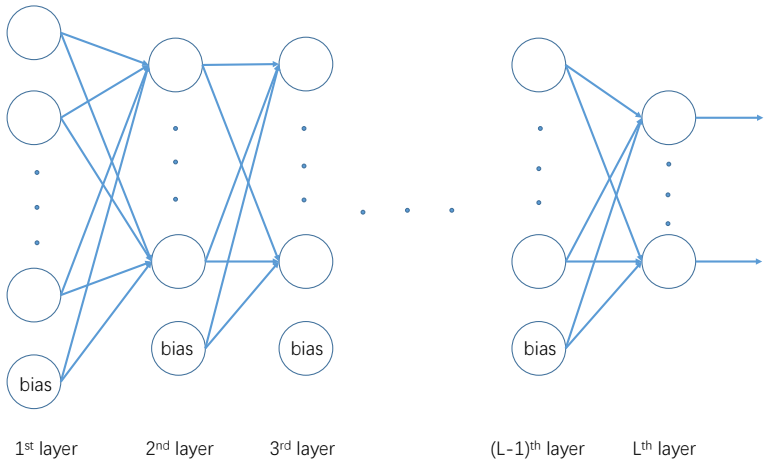


Fig. 2.1. General structure of feed-forward neural networks

As shown in Fig. 2.1, the feed-forward neural network contains L layers. The first layer is the input layer, the last layer is the output layer and the others are hidden layers. The number of layers L is usually called the depth of the neural network. Denote N_l the number of units in l-th layer. A bias term, also called threshold, is always added to each layer except the output layer. The biases are usually set to be one. Each layer will extract features from the layer before it, so that neural network is hierarchical in some sense.

Put it more specifically, the general framework of the feed-forward pass of a neural network can be described as follows. Let $f_1, ..., f_L$ be activation functions for each of the L layers. Then the activation rule can be described as

$$f_l^{W,b} := f_l(\sum_{j=1}^{N_l} W_{lj} X_j + b_l) = f_l(W_l X_l + b_l), \qquad 1 \leq l \leq L,$$

where b_l can be regarded as the bias or the weight of bias.

Let $a^{(l)}$ denote the activation levels or activation values of l-th layer. It is common to set $a^{(1)} = X$. The explicit structure of the activation rule or

the feed-forward pass is then

$$
\begin{aligned}
z^{(2)} &= W^{(1)}X + b^{(1)}, \\
a^{(2)} &= f^{(1)}(z^{(2)}), \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)}, \\
a^{(3)} &= f^{(2)}(z^{(3)}), \\
&\cdots \\
z^{(L)} &= W^{(L-1)}a^{(L-1)} + b^{(L-1)}, \\
a^{(L)} &= f^{(L-1)}(z^{(L)}).
\end{aligned}
$$

Here, $W^{(l)}$ are weight matrices and $b^{(l)}$ are biases. Specifically, $W^{(l)} \in R^{N_l \times N_{l-1}}$ and $b^{(l)} \in R^{N_{l+1}}$

The choice of activation function $f^{(l)}$ has a great impact on the performance of neural networks. Commonly used deterministic activation functions are linear function, sigmoid functions (e.g., $1/(1+exp(-x))$, $tanh(x)$), or rectified linear units (ReLU) $max\{x, 0\}$. ReLU is very popular in machine learning nowadays since it can reduce the likelihood of the gradient to vanish and it is sparse so that training a neural network when ReLU is used would be faster. Through this study, we used ReLU as the activation function. The main advantage of ReLU in this study is that it has similar structure with the profit of option.

2.2 K-fold cross validation

Before training a neural network, it is common to divide the dataset into three subsets, namely training set, validation set and testing set. The training set is used to train the weights of the neural network. The validation

set is used to reduce overfitting and tune the hyperparameters. Finally, the testing set is used to test the actual performance of the trained neural network.

However, sometimes we will use a technique named k-fold cross validation when the size of the dataset is not big enough. The technique will also make the results more stable. By k-fold cross validation, we will split the original dataset into training set and testing set, without a validation set. Then we will split the training set into k subsets of equal size to conduct training and validation on different subsets. The cross validation process will repeat k times. Each time a single subset is left as the validation set to test the model and all the other $k-1$ subsets are used to train the model. For example, we are going to conduct a 4-fold cross validation on a dataset of 1000 samples. Then the dataset will be split into 4 subsets, each of which will contain 250 samples. The process will be repeated 4 times and each time one subset will be left as the validation set and all the other three subsets will be used to train the model. The process is shown in Fig. 2.2.

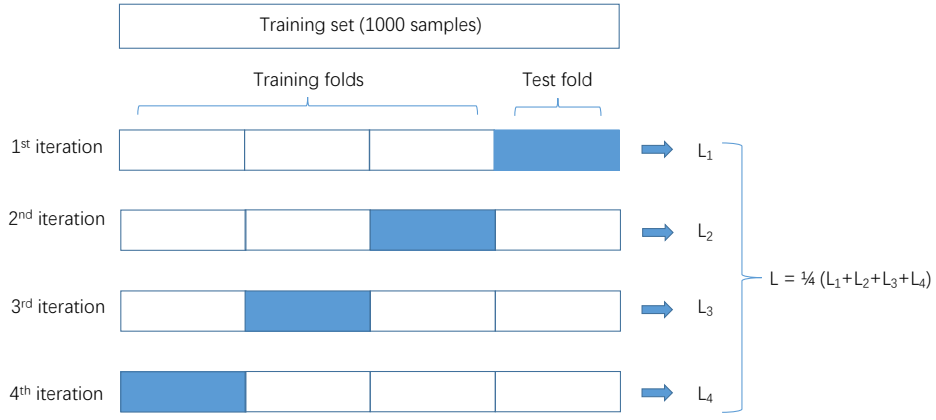


Fig. 2.2. The process of 4-fold cross validation

By the cross validation process, we can also minimize the overfitting and increase the performance on the test set of the trained model.

Usually the subsets are randomly partitioned, however, when dealing with time series, it is common to divide the training data into disjoint time periods of identical length. In this study, we use 4-fold cross validation.

2.3 Training a neural network

Once the number of hidden layers, the number of units in each hidden layer and the activation function have been chosen, we need to find $(\hat{W}, \hat{b})$, where

$$\hat{W} = (\hat{W}^{(1)}, ...\hat{W}^{(L-1)}) \qquad and \qquad \hat{b} = (\hat{b}^{(1)}, ..., \hat{b}^{(L-1)})$$

denote the learning parameters which are computed during training.

Let $D = \{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), ..., (X^{(T)}, Y^{(T)})\}$ be the training dataset which contains T input-output pairs. Let $L(Y, \hat{Y})$ be the loss function where Y is the actual target value and $\hat{Y}$ is the predicted output. Then the problem we need to solve is

$$argmin_{W,b} \frac{1}{T} \sum_{i=1}^{T} L(Y^{(i)}, \hat{Y}^{W,b}(X^{(i)})). \tag{2.1}$$

Often, the loss is measured by L_2-norm, that is

$$L(Y^{(i)}, \hat{Y}(X^{(i)})) = \|Y^{(i)} - \hat{Y}(X^{(i)})\|_2^2.$$

Then our target function (2.1) becomes the mean-squared error (MSE) over

the training set.

Overfitting is a common problem in the process of training a neural network. In that case, the out-of-sample performance is not as good as the performance over the training dataset. To reduce it, a regularization term, denoted by $\phi(W, b)$, is often added to our target function. Adding a regularization penalty is actually a tradeoff between bias and variance. We then need to solve

$$argmin_{W,b}\frac{1}{T}\sum_{i=1}^{T} L(Y^{(i)}, \hat{Y}^{W,b}(X^{(i)})) + \lambda\phi(W, b). \tag{2.2}$$

The amount of regularization, λ, will also have a great impact on the performance of the model. Too little regularization will still cause over-fitting and too much will lead to under-fitting. We choose λ by cross validation.

We will take a separable penalty in many cases, that is, $\phi(W, b) = \phi(W) + \phi(b)$. However, applying a regularization term to the bias units usually makes only a small difference. Thus the most often used penalty is

$$\phi(W) = \|W\|_2^2 = \sum_{i=1}^{T} W_i^T W_i,$$

which can be viewed as a default choice.

Dropout is also a technique for reducing overfitting. We will discuss it in Section 2.4.

2.3.1 Back-propagation

The commonly method to solve (2.2) is of a form of gradient descent. The batch gradient descent (BGD) computes the gradient using the whole

dataset while the stochastic gradient descent (SGD) using a single sample. Often, the SGD converges much faster than BGD, but it will not minimize the error function as well as the BGD. The mini-batch gradient descent (MBGD) makes a compromise between the two methods, see LeCun et al (2012) [5]. It updates the gradient descent using a few samples. The batch size is decided manually. In practice it is sometimes difficult to choose an appropriate learning rate when using gradient descent methods. Adagrad [6], Adadelta [7] and Adam [8] are all algorithms based on gradient descent method which can get adaptive learning rates for different parameters. In this study, we use Adam as the optimization method.

No matter what kind of advance optimization method is used, it has to be combined with the back-propagation algorithm. Back-propagation is an algorithm which provides an efficient way to get the first-order partial derivatives needed in the process of training a neural network.

For example, after one iteration of gradient descent, the parameters are updated as follows:

$$\begin{aligned} W_{i,j}^{(l)} &= W_{i,j}^{(l)} - \alpha \frac{\partial}{\partial W_{i,j}^{(l)}} J(W,b), \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b), \end{aligned} \tag{2.3}$$

where α is the learning rate and $J(W,b)$ denotes the target function in (2.2). Back-propagation algorithm is used to compute the derivatives $\frac{\partial}{\partial W_{i,j}^{(l)}} L(Y,\hat{Y}(X))$ and $\frac{\partial}{\partial b_i^{(l)}} L(Y,\hat{Y}(X))$. With $\frac{\partial}{\partial W_{i,j}^{(l)}} L(Y,\hat{Y}(X))$ and $\frac{\partial}{\partial b_i^{(l)}} L(Y,\hat{Y}(X))$ known, we can get the partial derivatives of $J(W,b)$ in (2.3). Thus, back-propagation algorithm is very important.

Since we only used neural networks with only one hidden layer through

this study, here we will explain the back-propagation algorithm with a simple three-layer neural network in detail. The structure of the three-layer neural network is shown in Fig. 2.3

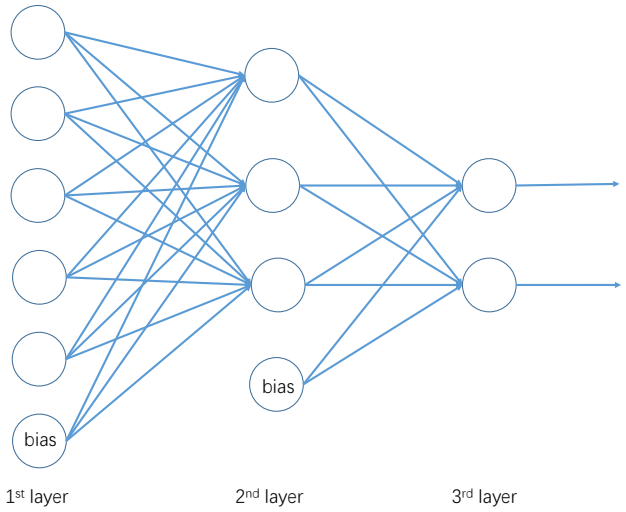


Fig. 2.3. The structure of a simple 3-layer neural network

As shown in Fig. 2.3, the input is a 5-dimensional vector, the output is a 2-dimensional vector, and the only hidden layer contains three hidden units. The neural network is fully connected. The explicit feed-forward pass of the neural network is

$$
\begin{aligned}
z^{(2)} &= W^{(1)}X + b^{(1)}, \\
a^{(2)} &= f_2(z^{(2)}), \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)}, \\
Y &= F^{W,b}(X) = a^{(3)} = f_3(z^{(3)}),
\end{aligned}
$$

where $X \in R^5$, $W^{(1)} \in R^{3\times5}$, $W^{(2)} \in R^{2\times3}$, $b^{(1)} \in R^3$, $b^{(2)} \in R^2$ and $Y \in R^2$.

The first step of back-propagation algorithm is to perform the feed-forward pass and compute the activation levels for layer 2 and layer 3 using the equations in (2.4). Then we get the values of $a^{(2)}$ and $a^{(3)}$.

The second step is to compute

$$\delta^{(3)} = -(Y - a^{(3)}).$$

Here $\delta^{(3)}$ can be regarded as the error between the output and the actual target value.

The third step is to compute

$$\delta^{(2)} = ((W^{(2)})^T) \star f\prime(z^{(2)}),$$

where $\star$ denotes element-wise product and $f\prime$ is the first-order derivative of the activation function. In this study, the first-order derivative of ReLU is

$$f\prime(x) = \begin{cases} 1 & \text{if x is bigger than 0} \\ 0 & \text{if x is not bigger than 0} \end{cases}$$

If the neural network has L layers, then we need to compute $\delta^{(l)}$, for $l = L-1, L-2, ..., 2$. Pay attention here that we do not have to compute $\delta^{(1)}$ for the input layer since the input layer does not have an error.

The last step is to compute the desired partial derivatives:

$$\bigtriangledown_{W^{(l)}} L(Y, \hat{Y}(X)) = \delta^{(l+1)}(a^{(l)})^T,$$
$$\bigtriangledown_{b^{(l)}} L(Y, \hat{Y}(X)) = \delta^{(l+1)},$$

where $l = 1, 2$. Then we get the partial derivatives we want.

2.4 Dropout for model selection

Dropout is a regularization technique which was raised by Hinton et al (2012) [9]. It is designed to avoid over-fitting by decreasing input dimensions in X randomly with a given probability p. For example, in a model with one hidden layer, the original neural network is

$$a^{(l)} = f(z^{(l)}),$$
$$z^{(l)} = W^{(l)}X + b^{(l)},$$

and the network with dropout architecture is

$$D^{(l)} \sim Ber(p),$$
$$\tilde{a}^{(l)} = D^{(l)} \star X,$$
$$a^{(l)} = f(z^{(l)}),$$
$$z^{(l)} = W^{(l)}X + b^{(l)},$$

where D is a matrix of independent random variables which follow the Bernoulli $Ber(p)$ distribution.
In a neural network with dropout, one or more units in the hidden layer is switched off once in a while so that it will not interact with the network. The learned weights of the units become somewhat less sensitive to the weights of the other units. In general, dropout helps the neural network to generalize better and increase accuracy since the influence of a small set of input data is decreased by dropout.

The probability p is usually set between 0.2 to 0.5. In this study, we set $p = 0.2$.

2.5 Auto-encoder

Auto-encoder is a special type of neural network. It tries to train the architecture to approximate X by itself, which means that $Y = X$. In that way, we try to find the model $F^{w,b}(X)$ which can recreate X by the concentrated information. In other words, an auto-encoder creates a more cost effective representation of the input X.

The structure of an auto-encoder in this study can be shown in Fig. 2.4. As shown in Fig. 2.4, it is a bottleneck structure. The output layer and the input layer both have ten units. The hidden layer contains five units. In this study the 10-dimensional input vector represents the prices of ten stocks at the same time. The hidden layer compresses the aggregate information contained in the ten stocks and then produces an approximation of everyinput stock. It gives a more efficient way to represent stock prices and the stock market.

Put it more specifically, the explicit feed-forward pass of the auto-encoder is

$$
\begin{aligned}
z^{(2)} &= W^{(1)}X + b^{(1)}, \\
a^{(2)} &= f_2(z^{(2)}), \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)}, \\
Y &= F^{W,b}(X) = a^{(3)} = f_3(z^{(3)}),
\end{aligned}
$$

where $a^{(2)}$, $a^{(3)}$ are activation levels. The goal is to learn the weight matrices $W^{(1)}$, $W^{(2)}$, which solve

$$argmin_{W,b}\|F^{W,b}(X) - X\|_2^2$$

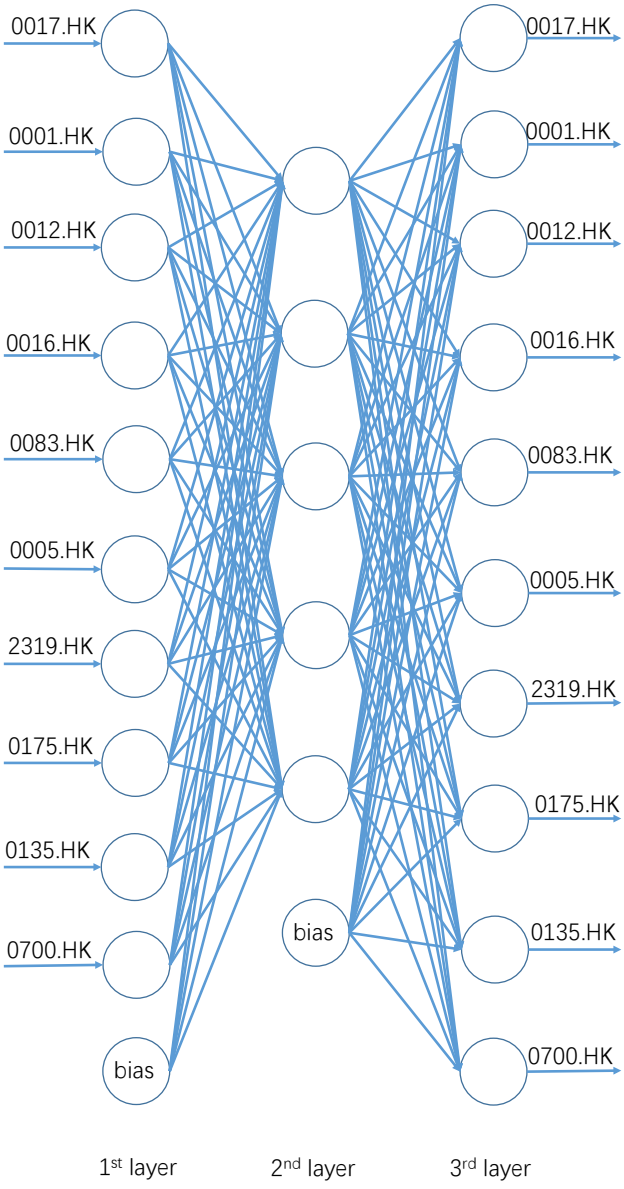


Fig. 2.4. The structure of auto-encoder

subject to a regularization penalty on the weights.

Chapter 3

Constructing Portfolios

A financial portfolio is a group of financial assets, which can be represented as

$$y = X\omega + b,$$

where X is a matrix represents returns of different assets and b, ω represent the risk free rate and the portfolio weights.

3.1 Markowitz's modern portfolio theory

Modern portfolio theory was raised by Markowitz in 1952 [1]. It seeks to find the portfolio with the least risk under a fixed return or with the largest return under a fixed risk. It is actually a tradeoff between the mean (return) and the variance (risk).

Consider a large amount of input data $X = (X_{i,t})_{i,t=1}^{N,T}$, where X is the return of stocks, N is the number of stocks and T is the number of time periods. In financial problems, X is always a skinny matrix since N is usually much smaller than T. For example for Hang Seng index (HSI) $N = 50$, and T can be very large depending on the trading intervals.

In Markowitz's approach, it computes the empirical mean and covariance matrix as follows:

$$\overline{X}_i = \frac{1}{T}\sum_{t=1}^{T} X_{i,t},$$
$$X_i X_i^T = \frac{1}{T}\sum_{t=1}^{T}(X_{i,t} - \overline{X}_i)(X_{i,t} - \overline{X}_i)^T.$$

The mean and the covariance matrix can be seemed as a compression of the information in the market. The original dataset is of size $N \times T$ and now the Markowitz approach has reduced it to size N for the means and $N(N-1)/2$ for the covariance matrix.

However, in practice the means and the covariance matrix are usually a poor representation of the information in the market. Using the historical mean to build the model will always cause large errors since it ignores all the periods of large jumps and volatility. The models based on Markowitz's approach are always sensitive to the inputs and thus lead to poor out-of-sample performance.

3.2 Black-Litterman model

Black-Litterman model seeks to overcome the problems encountered when applying modern portfolio theory.

Black-Litterman model will take investors' views into account. The mean in Black-Litterman model is no longer the empirical mean but a weighted average of the empirical mean and the estimated mean by investors. The more confident the investor is, the larger the weights of the

estimated mean. Put it simply, Black-Litterman model combines the prior information with the historical information.

3.3 Deep portfolio theory

3.3.1 Construction of deep portfolio

Deep portfolio theory (DPT) differs the traditional financial models in its deep architecture. Suppose that the data extracted from the market has been separated into a training set X and a validation set $\hat{X}$. Four steps are needed to construct a deep portfolio.

The first step is to encode the market. It solves the problem

$$min_W \|X - F_W^m(X)\|_2^2 \quad subject \quad to \quad \|W\| \leq L^m, \tag{3.1}$$

where X is the matrix of stock prices and $F_W^m(X)$ denotes the market-map. It creates a more information-efficient representation of X. Then we choose the stocks by the differences between the original stocks and their market-mapped version.

The second step is calibrating. It seeks to find the portfolio-map for a target Y, which is the price of stock index in this study. It solves the problem

$$min_W \|Y - F_W^p(X)\|_2^2 \quad subject \quad to \quad \|W\| \leq L^p, \tag{3.2}$$

where $F_W^p(X)$ denotes the portfolio-map and X is the price matrix of the stocks chosen from step one. Here $F_W^m(X)$ and $F_W^p(X)$ are independent.

The third step is validating. It is to find L^m and L^p which balance the

tradeoff between the two errors

$$\epsilon_m = \|\hat{X} - F^m_{W^*_m}(\hat{X})\|_2^2 \quad and \quad \epsilon_p = \|\hat{Y} - F^p_{W^*_p}(\hat{X})\|_2^2,$$

where W^*_m and W^*_p are the solutions of (3.1) and (3.2).

The last step is verification. We need to choose the market-map and the portfolio-map which satisfy the validating step by plotting the deep frontier.

3.3.2 Relationships between DPT and traditional financial models

As we had mentioned before, the Markowitz model may get poor out-of-sample performance in practice. However, this can be easily avoided by DPT. We can add a regularization term or use the dropout technique in the process of training the model. By varying the value of λ or adding a dropout layer, we can search over the architectures which not only fit the training set but also give good out-of-sample performance on the validation set.

In some sense, the Markowitz model and the Black-Litterman model can be regarded as the encoding step of DPT. However, the Markowitz model correspondes to the encoding step only determined by the empirical mean and variance-covariance matrix. And for Black-Litterman model, it gives a better auto-encoding of the market since it incorporates the investors' beliefs which can be considered as a form of regularization.

Chapter 4

Applications of beating the stock indices

In this study, we apply the DPT to design deep portfolios to track and beat the Hang Seng index (HSI) and Shanghai Stock Exchange (SSE) 180 index. We trained our models without knowledge of the weights of component stocks of the indices. Merton (1971) [10] raised the 1%-problem which is to find the strategy to beat a given benchmark by 1% per year. In this study, we try to find the portfolios which can beat stock indices when large breakdown happens by replacing all returns smaller than -5% to 5% in the calibration phase, see Heaton, Polson and White (2016) [3]. Then if the trained portfolios can beat the indices, it is of high probability to make profits by longing the portfolios and shorting the exchanged traded fund (ETF) of indices at the same time.

4.1 The datasets

For both HSI and SSE 180 index, we did three experiments. The three experiments went through the periods January 2012 to December 2016, January 2010 to December 2015 and January 2008 to December 2013, respectively. They are named experiment 1, experiment 2 and experiment 3,

respectively. We consider weekly prices here.

The normalization of data is necessary. In all the experiments in this study, we divide the prices of every stock by the price of the first day in the experiment. For example, if the prices of one stock are \$33.69, \$33.05, \$32.11, \$27.64, \$29.24, ..., the data we feed to our model is $\frac{\$33.69}{\$33.69}, \frac{\$33.05}{\$33.69}, \frac{\$33.11}{\$33.69}, \frac{\$27.64}{\$33.69}, \frac{\$29.24}{\$33.69}, ...$, that is, $1.000, 0.981, 0.983, 0.820, 0.868, ...$ The normalization in this study has two advantages. One is that the ranges of the relative prices for different stocks will not differ too much. Another advantage is that the relative prices can also reflect whether one stock has gone through large fluctuation.

4.2 Results for HSI

The HSI is an important indicator of the average market performance in Hong Kong. It consists of stocks from 50 different companies.

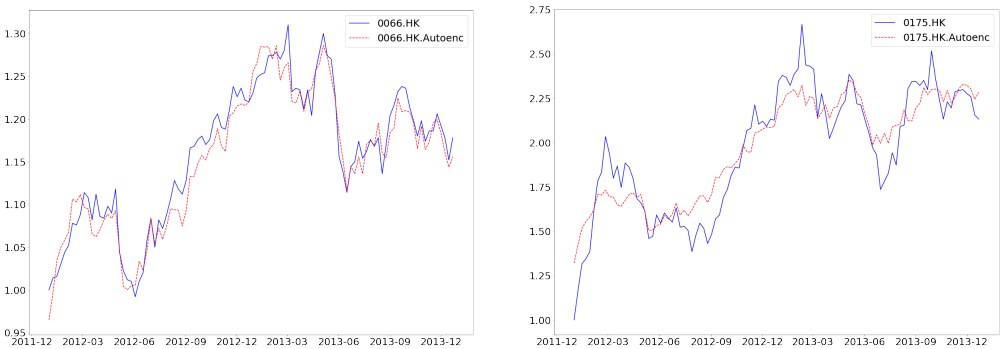
Here we will only show the results for the period January 2012 to December 2016. And more results will be shown in Section 4.4.

4.2.1 Smart indexing the HSI

For the four steps in constructing a deep portfolio, we conduct auto-encoding and calibration on the period January 2012 to December 2013, and validation and verification on the period January 2014 to December 2015. We used one hidden layer with five units for all neural networks.

After auto-encoding all the component stocks of HSI, we rank all the 50 stocks by the 2-norm difference between every stock and its auto-encoded

version. The smaller the difference, the higher the communal information content of the stock. We want to identify a small group of stocks which can congregate a large part of the total information of the universal stocks rather than a small group of stocks which have given similar performances with the index. In the process of reproducing the universe of stocks from a bottleneck structure, the auto-encoder reduces the total information of the market to a subset of information which is applicable to a large number of stocks. Therefore, the difference of a stock between its auto-encoded version provides a measure for the similarity of a stock with the stock universe. The closer a stock to its auto-encoded version, the more similarity of the stock with the stock universe. Bearing this idea, we construct the deep portfolio by 5 most communal stocks and x-number of most non-communal stocks ($x = 5, 10, 15, 20, 25$). For example, 15 stocks means 5 most communal stocks plus 10 most non-communal stocks. Stocks 0066.HK (MTR Corporation Limited) and 0175.HK (Geely Automobile Holdings Ltd) are the stocks with the highest and lowest communal information, respectively. In Fig. 4.1, we see the two stocks with their auto-encoded version.



(a) (b)

Fig. 4.1. Auto-encoding for HSI in experiment 1: most communal/non-communal examples

In Fig. 4.2, we present the calibration results for portfolios with 10, 20,

and 30 stocks.

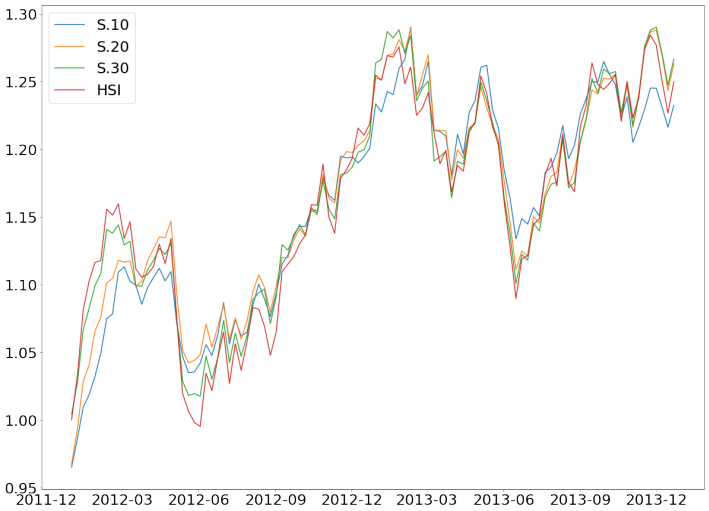


Fig. 4.2. Calibration step for HSI in experiment 1

In Fig. 4.3, we see the validation results for different portfolios. We used the models from the calibration step to track the HSI for the period January 2014 to December 2015. We can see that the portfolios with 20 and 30 stocks track the index quite well.

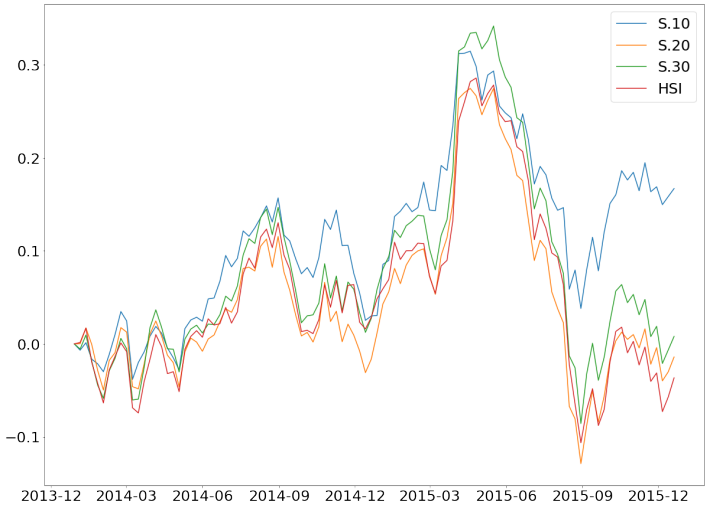


Fig. 4.3. Validation step for HSI in experiment 1

Fig. 4.4 shows the deep frontier. We need to make a tradeoff between the number of stocks used to construct portfolio and the validation error.

From Fig. 4.4 we see that the deep portfolio with 20 stocks is enough to track the HSI.

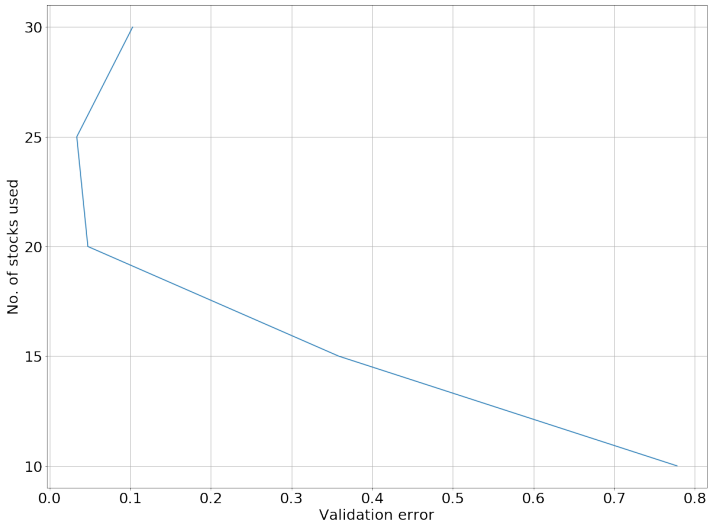


Fig. 4.4. Deep frontier for HSI in experiment 1

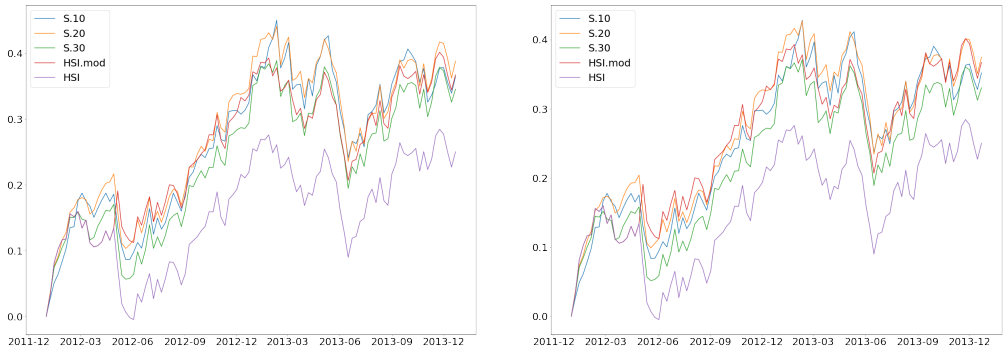
4.2.2 Beating the HSI

In order to beat the HSI, we altered the prices of HSI during the calibration step by replacing all returns smaller than -5% to 5%. Then we proceeded exactly as the process in Section 4.2.1. And in this section, we applied both neural networks without dropout and with dropout.

Fig. 4.5 shows the results of the calibration step. Fig. 4.5a and Fig. 4.5b show the successful replication of the modified HSI by different portfolios without and with dropout, respectively.

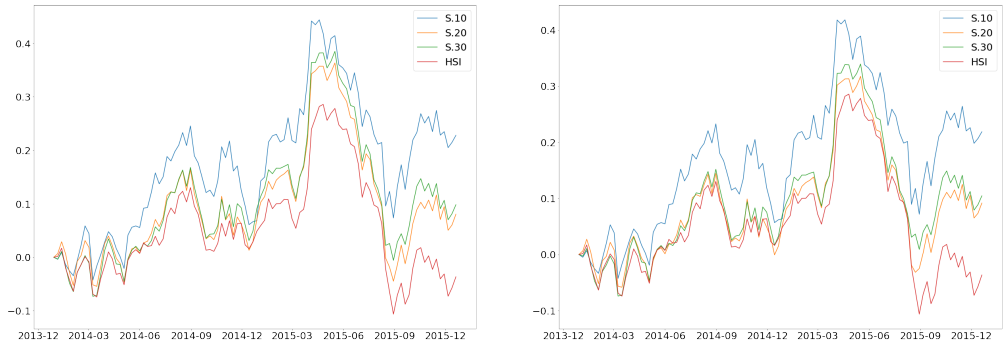
Fig. 4.6 shows the results of the validation step. Compared with Fig. 4.3, the portfolios in Fig. 4.6 achieve outperformance in some sense.

Fig. 4.7 shows the deep frontiers. We can see that a portfolio with 15 stocks is enough for both neural networks. And the neural network with dropout gets smaller validation errors than the neural network without



(a) (b)

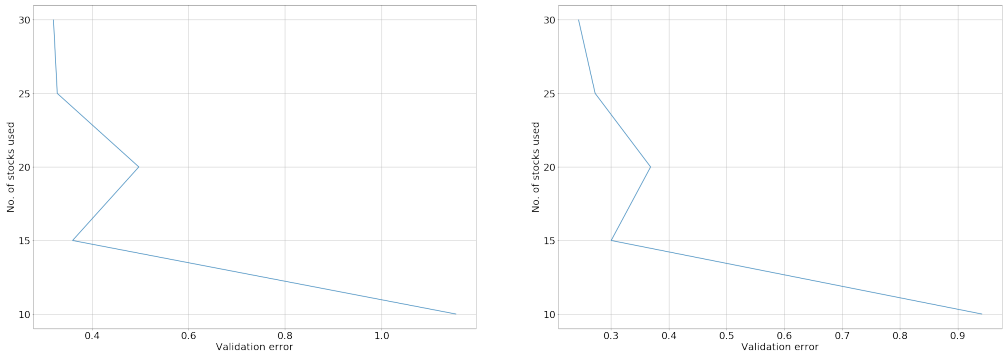
Fig. 4.5. Calibration step for beating HSI in experiment 1: (a) without dropout; (b) with dropout



(a) (b)

Fig. 4.6. Validation step for beating HSI in experiment 1: (a) without dropout; (b) with dropout

dropout.



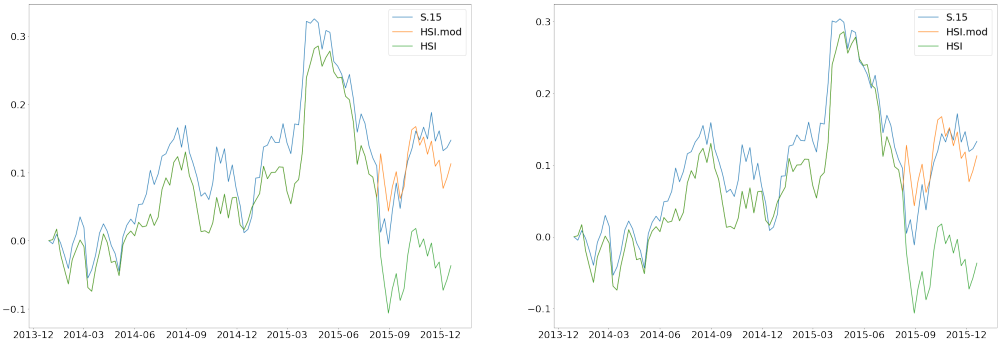
(a) (b)

Fig. 4.7. Deep frontiers for beating HSI in experiment 1: (a) without dropout; (b) with dropout

4.2.3 A realistic implementation

The results in Section 4.2.2 show that the deep portfolio can beat the stock index. In this section, we try to make it a realistic implementation. The expected profit is the difference between the price of the portfolio and the price of the stock index.

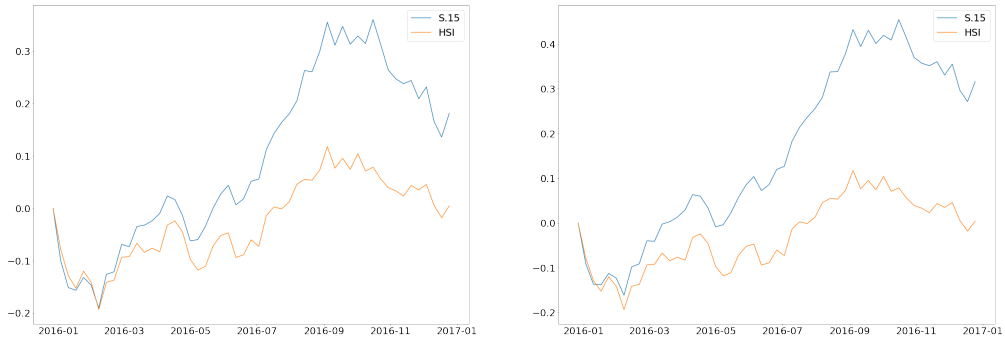
First we train the model using the data from January 2014 to December 2015, as shown in Fig. 4.8 .



(a) (b)

Fig. 4.8. Constructing portfolios for beating HSI in experiment 1: (a) without dropout; (b) with dropout

Then we try to construct the deep portfolios which have the same behavior with the modified index. Then by longing the portfolio and shorting the ETF of HSI for the same period, it will give a great probability to make profits. Actually, every unit in the hidden layer of the neural network can be regarded as a basket call option such that the deep portfolio is a linear combination of basket options. So the strategy is to long the deep portfolio and short the ETF of HSI for the next one year. The ideal result is shown in Fig. 4.9 . And in a real implementation, the profits are 14.74% and 18.04%, respectively. Here we ignore the prices of options, so the real profits will be smaller than the profits above.



(a) (b)

Fig. 4.9. Ideal profits of the portfolios for beating HSI in experiment 1: (a) without dropout; (b) with dropout

4.3 Results for SSE 180 index

SSE 180 index is the stock index of Shanghai Stock Exchange. It is the most commonly used indicator to reflect Shanghai Stock Exchange's market performance.

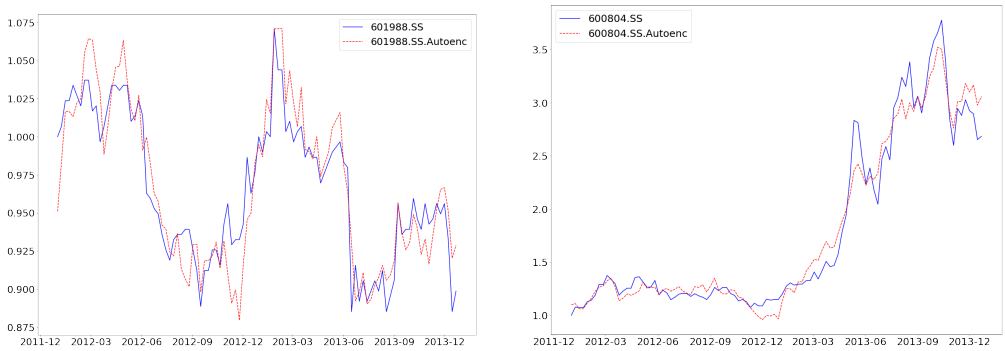
Here we will only show the results for the period January 2012 to December 2016. And more results will be shown in Section 4.4. We did smart indexing and outperforming as it exactly in Section 4.2.1 and Section 4.2.2.

4.3.1 Smart indexing the SSE 180 index

For the four steps in constructing a deep portfolio, we also conduct auto-encoding and calibration for the period January 2012 to December 2013, and validation and verification for the period January 2014 to December 2015.

Stocks 601988.SS (Bank of China Ltd) and 600804.SS (Dr Peng Telecom& Media Group Co Ltd) are the stocks with the highest and lowest communal information, respectively. Fig. 4.14 shows the two stocks with their auto-encoded version. After auto-encoding, we construct the deep portfolios by 10 most communal stocks and x-number of most non-communal

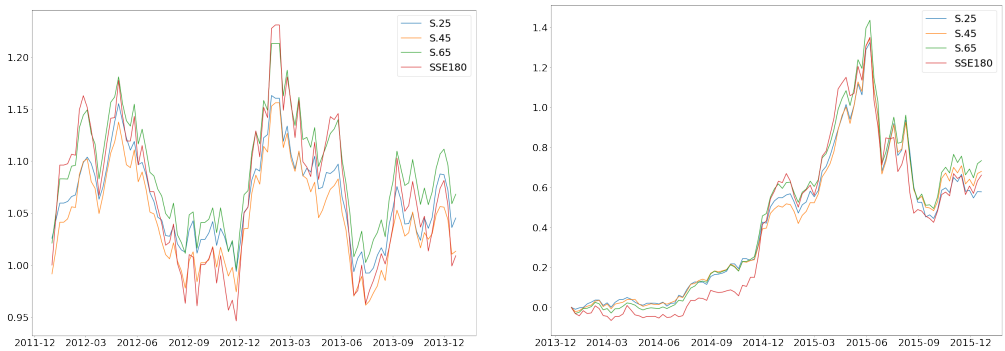
stocks (x=10,15,20,25,30,35,40,45,50,55).



(a) (b)

Fig. 4.10. Auto-encoding for SSE 180 index in experiment 1: most communal/non-communal examples

In Fig. 4.11a, we present the calibration results for portfolios with 25, 45, and 65 stocks. In Fig. 4.11b, we see the validation results for different portfolios. We used the models from the calibration step to track the SSE 180 index for the period January 2014 to December 2015.



(a) (b)

Fig. 4.11. Calibration step and validation step for SSE 180 index in experiment 1

Fig. 4.12 shows the deep frontier. We need to make a tradeoff between the number of stocks and the validation error. From Fig. 4.12 we see that the deep portfolio with 40 stocks is enough to track the SSE 180 index.

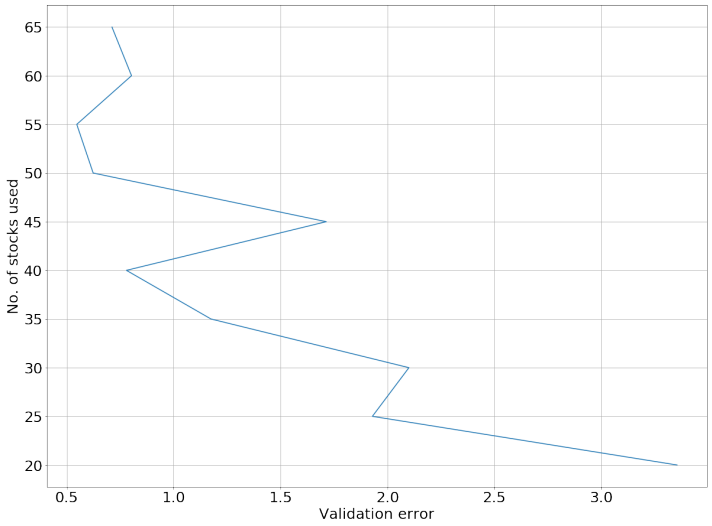
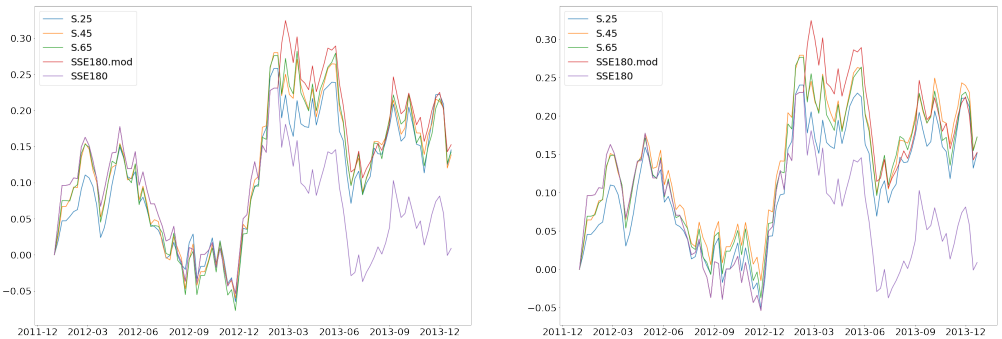


Fig. 4.12. Deep frontier for SSE 180 in experiment 1

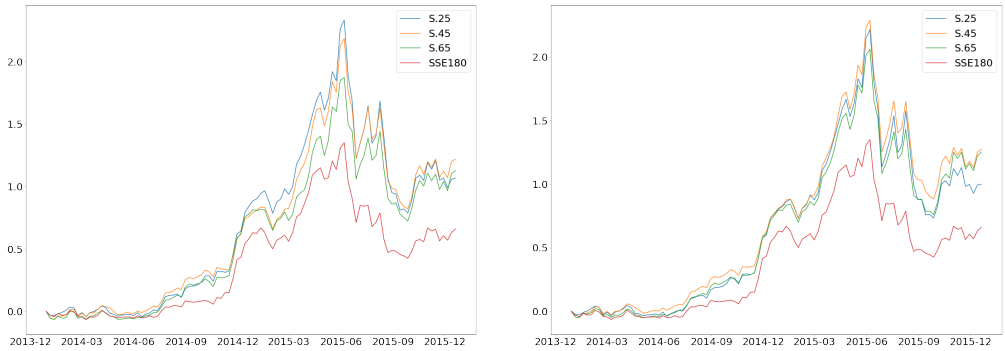
4.3.2 Beating the SSE 180 index

In order to beat the SSE 180 index, we also altered the prices of SSE 180 index during the calibration step by replacing all returns smaller than -5% to 5%. Fig. 4.13 shows the successful replication of the modified SSE 180 index. Fig. 4.14 shows the results of the validation step. Compared with Fig. 4.11b, the portfolios in Fig. 4.14 achieve outperformance in some sense. Fig. 4.15 shows the deep frontiers. We observe that a portfolio with 40 stocks is enough to beat the index for both neural networks. And the neural network with dropout gets smaller validation errors.



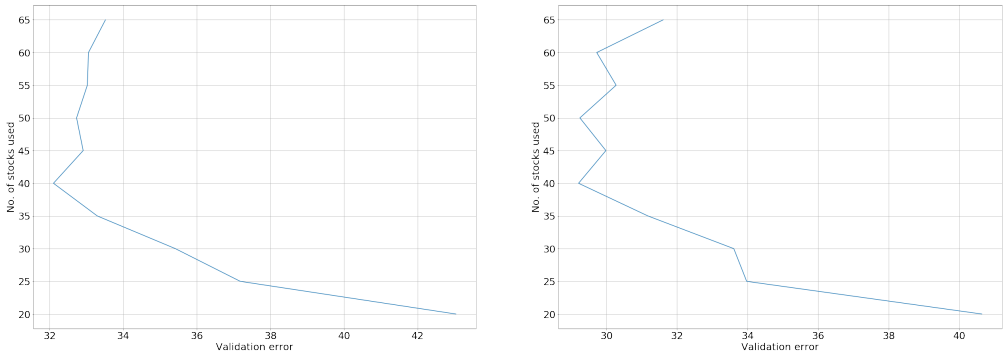
(a) (b)

Fig. 4.13. Calibration step for beating SSE 180 index in experiment 1: (a) without dropout; (b) with dropout



(a) (b)

Fig. 4.14. Validation step for beating SSE 180 index in experiment 1: (a) without dropout; (b) with dropout

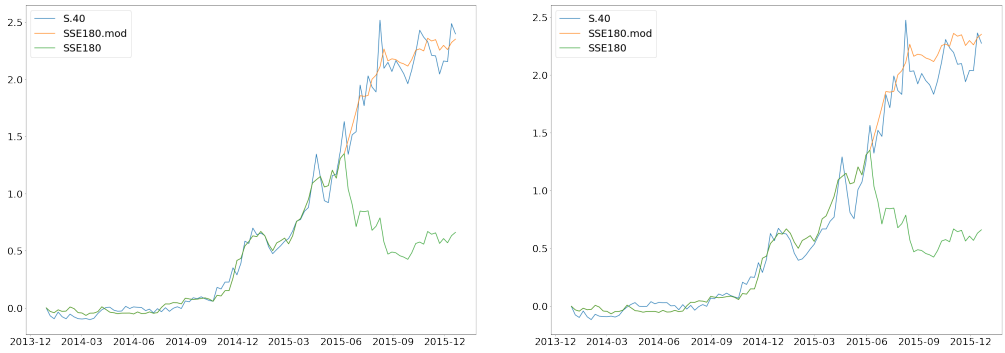


(a) (b)

Fig. 4.15. Deep frontiers for beating SSE 180 index in experiment 1: (a) without dropout; (b) with dropout

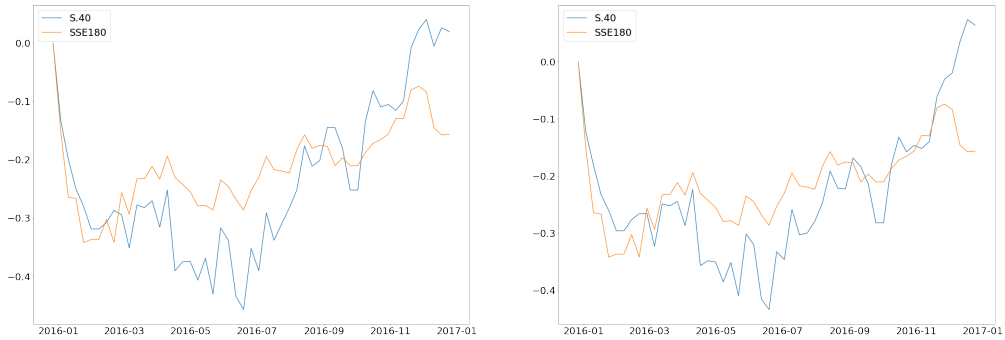
4.3.3 A realistic implementation

First we train the model using the data from January 2014 to December 2015, as shown in Fig. 4.16 . Then we try to construct the deep portfolios which have the same behavior with the modified index. Then by longing the portfolio and shorting the ETF of SSE 180 index for the next one year, it will give a great probability to make profits. The ideal result is shown in Fig. 4.17 . And in a real implementation, the profits are 5.57% and 6.98%, respectively.



(a) (b)

Fig. 4.16. Constructing portfolios for beating SSE 180 index in experiment 1: (a) without dropout; (b) with dropout



(a) (b)

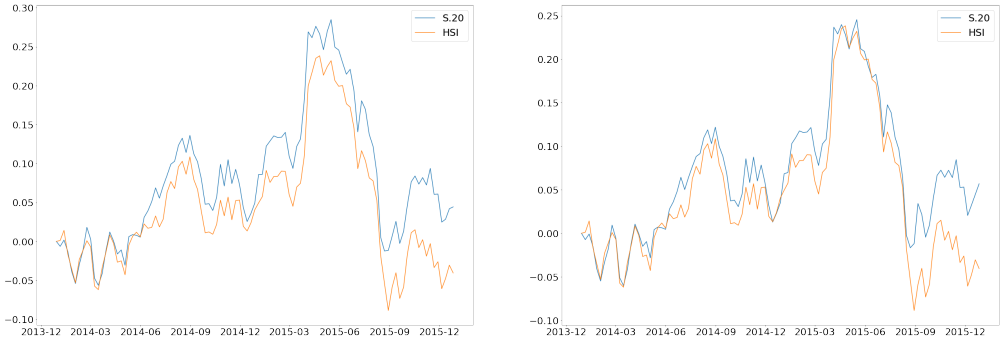
Fig. 4.17. Ideal profits of the portfolios for beating SSE 180 index in experiment 1: (a) without dropout; (b) with dropout

4.4 More results

4.4.1 More results for beating HSI

Fig. 4.18 shows the results for beating HSI in experiment 2. It shows that our portfolios beat the HSI when we altered the prices of HSI for the period January 2012 to December 2013 by replacing all returns smaller than -5% to 5%. If we hold the portfolio for one year from January 2014 to December 2014, the returns are 0.48% and -2.31%, respectively. If we hold the portfolio for two years from January 2014 to December 2015, the returns are 7.56% and 9.70%, respectively.

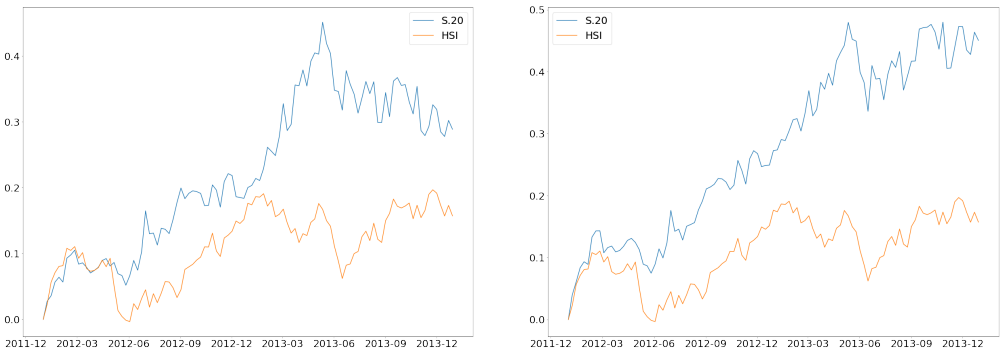
Fig. 4.19 shows the results for beating HSI in experiment 3. It shows



(a) (b)

Fig. 4.18. Ideal profits of the portfolios for beating HSI in experiment 2: (a) without dropout; (b) with dropout

that our portfolios beat the HSI when we altered the prices of HSI for the period January 2010 to December 2011 by replacing all returns smaller than -5% to 5%. If we hold the portfolio for one year, the returns are 5.28% and 13.27%, respectively. If we hold the portfolio for two years, the returns are 10.53% and 23.48%, respectively.



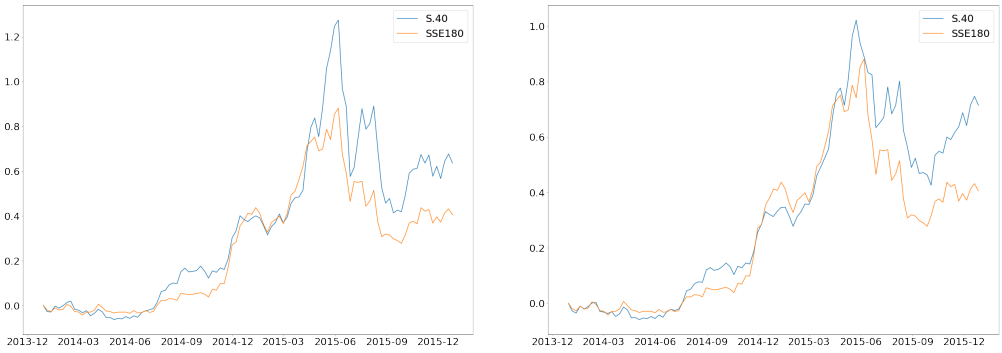
(a) (b)

Fig. 4.19. Ideal profits of the portfolios for beating HSI in experiment 3: (a) without dropout; (b) with dropout

4.4.2 More results for beating SSE 180 index

Fig. 4.20 shows the results for beating SSE 180 index in experiment 2. It shows that our portfolios beat the SSE 180 index when we altered the prices of SSE 180 index for the period January 2012 to December 2013

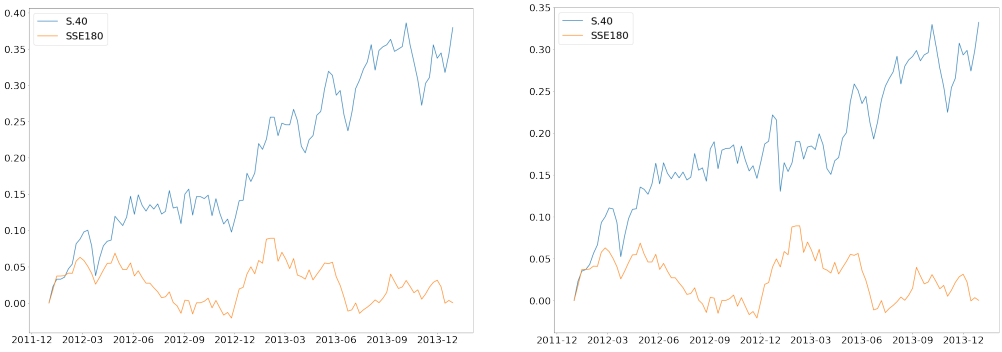
by replacing all returns smaller than -5% to 5%. If we hold the portfolio for one year, the returns are -3.05% and -8.54%, respectively. If we hold the portfolio for two years, the returns are 20.03% and 26.85%, respectively.



(a) (b)

Fig. 4.20. Ideal profits of the portfolios for beating SSE 180 index in experiment 2: (a) without dropout; (b) with dropout

Fig. 4.21 shows the results for beating SSE 180 index in experiment 3. It shows that our portfolios beat the SSE 180 index when we altered the prices of SSE 180 index for the period January 2010 to December 2011 by replacing all returns smaller than -5% to 5%. If we hold the portfolio for one year, the returns are 19.03% and 26.54%, respectively. If we hold the portfolio for two years, the returns are 65.93% and 57.68%, respectively.



(a) (b)

Fig. 4.21. Ideal profits of the portfolios for beating SSE 180 index in experiment 3: (a) without dropout; (b) with dropout

Chapter 5

Conclusions

This study is based on the deep portfolio theory (DPT) raised by Heaton, Polson and White. DPT differs other standard financial models in its deep architecture. The key idea of DPT is to get a more efficient representation of the market by auto-encoding the universal stocks. The advantage of using neural networks is that overfitting is more easily reduced. After auto-encoding the universal stocks, we construct our portfolios to track and beat the stock indices by a specified level. Then we can make profits by longing the portfolios and shorting the exchanged traded fund (ETF) of the indices at the same time.

We have conducted three experiments of different periods on both the Hang Seng index (HSI) and the Shanghai Stock Exchange (SSE) 180 index. The results show that the neural network works well on both stock indices. The deep portfolios we constructed can beat the stock indices and we can always make profits by our strategies. Holding the portfolios for two years always makes more profits than holding for only one year since the prices of indices may not go through large breakdowns in a short period. And the portfolios using dropout are always making more profits than portfolios without using dropout. It proves that dropout can help improve the

performance of the models in this study.

In this study, we fixed the architecture of the neural networks to be of one hidden layer with five units. Actually we also tried neural networks with two hidden layers, but it did not improve the performance of our portfolios. The architectures are not so deep here, but it may need deeper architectures in other applications. We may also try other architectures, for example, to increase the number of units in the hidden layers. We may also try to apply the idea of deep architectures to other financial applications in the future.

Bibliography

[1] H. Markowitz, "Portfolio selection," *The journal of finance*, vol. 7, no. 1, pp. 77–91, 1952.

[2] F. Black and R. B. Litterman, "Asset allocation: combining investor views with market equilibrium," *The Journal of Fixed Income*, vol. 1, no. 2, pp. 7–18, 1991.

[3] J. Heaton, N. Polson, and J. Witte, "Deep portfolio theory," *arXiv preprint arXiv:1605.07230*, 2016.

[4] J. B. Heaton, N. G. Polson, and J. H. Witte, "Deep learning in finance," *CoRR*, vol. abs/1602.06561, 2016. [Online]. Available: http://arxiv.org/abs/1602.06561

[5] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[6] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[7] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv e-prints*, July 2012.

[10] R. C. Merton, "An analytic derivation of the efficient portfolio frontier," *Journal of financial and quantitative analysis*, vol. 7, no. 4, pp. 1851–1872, 1972.