**Nguyen Thi Huong Thu**

# BEATING THE INDEX WITH DEEP LEARNING

## A Method for Passive Investing and Systematic Active Investing

Master's Thesis

Oulu Business School

May 2021

UNIVERSITY OF OULU
Oulu Business School

ABSTRACT OF THE MASTER'S THESIS

| Unit | | | |
|---|---|---|---|
| Department of Finance | | | |
| Author | | Supervisor | |
| Nguyen Thi Huong Thu | | Conlin A., Postdoctoral researcher | |
| Title | | | |
| Beating the Index with Deep Learning: A Method for Passive Investing and Systematic Active Investing | | | |
| Subject | Type of the degree | Time of publication | Number of pages |
| Finance | Master's Thesis | May 2021 | 87 |


Abstract

In index tracking, while the full replication requires holding all the asset constituents of the index in the tracking portfolio, the sampling approach attempts to construct a tracking portfolio with a subset of assets. Thus, sampling seems to be the approach of choice when considering the flexibility and transaction costs. Two problems that need to be solved to implement the sampling approach are asset selection and asset weighting. This study proposes a framework implemented in two stages: first selecting the assets and then determining asset components' weights. This study uses a deep autoencoder model for stock selection. The study then applies the $L_2$ regularization technique to set up a quadratic programming problem to determine investment weights of stock components.

Since the tracking portfolio tends to underperform the market index after taking management costs into accounts, the portfolio that can generate the excess returns over the index (index beating) brings more competitive advantages to passive fund managers. Thus, the proposed framework attempts to construct a portfolio with a small number of stocks that can both follow the market trends and generate excess returns over the market index.

The framework successfully constructed a portfolio with ten stocks beating the S&P 500 index in any given 1-year period with a justifiable risk level.



Keywords
indexing, portfolio, sampling, autoencoder, regularization

Additional information

**CONTENTS**

## FIGURES

## TABLES

# 1 INTRODUCTION

Fund managers implementing the active investing strategy attempt to select the most attractive assets in the portfolio based on their deep analysis and expertise (Beasley, Meade & Chang, 2003). They also need to decide the right time to sell and buy those assets to "beating" the market. The actively managed fund beats the market when the fund's returns are higher than the benchmark index's returns. Rather than beating the market, passive investing attempts to achieve the same returns of the market index i.e. tracking the market index (Beasley et al., 2003). Passive fund managers do not buy and sell assets in the tracking portfolio as frequently as active counterparts; they may only rebalance the portfolio when the benchmark index is rebalanced or reconstituted.

Due to the high expenses spent for active management e.g. transaction costs from multiple long and short active positions, the management fees of actively managed funds are more expensive than that of passive ones. Moreover, the investment managers' decisions are not always accurate; their misjudgments can cause losses to the portfolio investment. Thus, although active investing is more flexible and its benchmark-outperforming target is tempting, the performance of active investing tends to be poorer than the benchmark index (Heaton, Polson & Witte, 2017a). The SPIVA US Scorecard published by S&P Dow Jones Indices in April 2020 reported that 71% of large-cap US funds failed to beat the S&P500 index in 2019. The performance of the actively managed funds becomes worse when it comes to a longer investment horizon. Particularly, 88.99% and 90.46% of large-cap US funds have underperformed the S&P500 index over a 10-year period and a 15-year period, respectively. Due to the unsatisfactory performance on actively managed funds, passive investing has obtained more investors' attention recently.

Two common approaches used to track the index are synthetic replication and physical replication (Maurer & Williams, 2015). The former focuses on derivative investing such as options and swaps, while the latter directly invests in assets. As synthetic replication involves contractual obligations, which contains the risk of defaulting (counterparty risks), physical replication is a more transparent method that investors can investigate the assets contained in the fund portfolio. Physical replication includes

two approaches that are full replication and sampling (Benidis, Feng & Palomar, 2018).

In the full replication approach, the fund managers hold all the index constituents in the tracking portfolio (Benidis et al., 2018). This results in huge transaction costs. For example, to track an equally weighted index, the tracking portfolio must be rebalanced frequently to keep the weight constant when there is a change in stock price, which is extremely costly if the tracking portfolio holds all the index constituents. The full replication method can be inflexible as some index constituents cannot be traded in the market due to their low liquidity. Moreover, the full replication method is inflexible when the structure of the benchmark index is not fully exposed; investors, therefore, cannot hold all the index constituents in their portfolios.

In contrast, the sampling approach is more flexible by only containing the most representative securities of the underlying index in the tracking portfolio (Benidis et al., 2018). Because only a subset of assets is selected, the sampling method can mitigate the trading costs when the tracking index is rebalanced or reconstituted. Although the sampling approach can generate large tracking errors, the sampling approach seems to be more promising than the full replication approach when considering the flexibility and transaction costs.

Although the passive management cost in the sampling approach is lower than the active one, the fact that index funds only produce the same returns as the index makes them underperform the market index by the number of management costs. Thus, using the sampling approach to construct a portfolio that can produce an excess return over market index (index beating) with bearable riskiness brings competitive advantages for passive fund managers.

Two problems that need to be solved when constructing the tracking portfolio with the sampling approach are how to select a subset of constituents of the underlying index (asset selection) and how much to invest in each of them (asset weighting). Some treat the two problems as a unified one (joint approach) while others solve them sequentially (two-step approach).

A lot of studies in the literature have introduced different methods for both asset selection and asset weighting. The methods of index tracking varied from the naïve strategy to the heavy computation. The development of machine learning and deep learning in the past few decades has significantly contributed to the evolution of index tracking methods. Brodie, Daubechies and De Mol (2009), Wu, Yang and Liu (2014), and Benidis, Feng and Palomar (2017) introduced a machine learning method called regularization, solving both asset selection and asset weighting problems at once. Focardi and Fabozzi (2014) applied the clustering method for asset selection typically.

To track the index, we may deal with the nonlinear interactions between the input and the output of the relevant data, which does not follow any financial theory (e.g., interactions of portfolio returns and index returns). Deep learning provides the type of model that can capture the nonlinear relationship between input and output. This type of model is called a neural network containing several layers; the model is trained in the way that the input data is gone and transformed through the layers of the neural network to map with the output (Lecun, Bengio & Hinton, 2015). Furthermore, the deep neural network is also applied to extract the features of data structure, which is promising for solving the stock selection problem. Thus, deep learning is a potential tool to deal with the problems in index tracking.

Heaton, Polson and Witte (2017b) pioneered in applying deep learning to build a framework for constructing a portfolio beating Biotechnology IBB Index. In their research, Heaton et al. (2017b) used an autoencoder for stock selection. Autoencoder is a special type of neural network where the output used in training is also its input (unsupervised learning method). Autoencoder is typically used in reducing the dimension and extracting the feature of the input. Thus, by using the auto-encoder, we can select stocks that share the most/least common information with the market, thereby creating a portfolio mimicking the market index. After obtaining the desired stocks from the autoencoder model, the paper again used neural networks to look for the relationship between portfolio returns and index returns. The market-beating strategy used in the research is to construct a portfolio with anti-correlations in the large drawdown periods. They created a modified index returns by replacing the original index's returns $\leq$ -5% by exactly 5%. Then they used the modified index returns to map with the portfolio returns in the training set, aiming to create a portfolio

specially generating returns higher than the index in the large drawdown of the market. Then they tested the out-of-sample performance of the constructed portfolio in the test set; the empirical results showed that their portfolio could beat the market index by 1% annually by containing at least 40 stocks.

Inspired by the success of Heaton et al. (2017b), this thesis applies the deep learning framework to construct a portfolio with a small number of stocks aiming to beat the market index. However, there are some differences between the framework used in this thesis and Heaton et al.'s:

- First, the benchmark index in this research is the S&P500 index instead of the Biotechnology IBB Index as in Heaton et al.'s.
- Second, while Heaton et al.'s study used the shallow architecture for the autoencoder model, the thesis designs the deeper architecture for the autoencoder network, which will be discussed in detail in the methodology section.
- Third, after the stock selection step, the thesis does not use the neural network to look for the relationship between the portfolio returns and index returns like Heaton et al.'s. The reason is that the neural networks require the data to transform through their multiple layers, which is not convenient for determining the direct effect between portfolio returns and index returns. Thus, determining invested weights of stock components is difficult in this approach. Alternatively, the thesis determines the invested weights by solving a quadratic programming problem capturing the direct relationship between portfolio returns and index returns. Additionally, the thesis applies the $L_2$ regularization method to enhance the out-of-sample performance of the portfolio.
- Fourth, the market-beating strategy used in this thesis is different from Heaton et al.'s. In Heaton et al.'s study, they attempted to beat the market by constructing a portfolio typically generating higher returns than the index in the large drawdown of the market. However, this required the portfolios to contain at least 40 stocks to have a reliable prediction. Unlike Heaton et al.'s study, the thesis does not attempt to construct a portfolio with anti-correlations in the large drawdown periods. Alternatively, the thesis proposes a new strategy for constructing a portfolio that can follow the market trends and

generally generates higher returns than the index. To do so, the thesis does not use the original index returns for model training, but index returns added 2%.

- Fifth, the thesis desires to construct a portfolio beating the market by containing only ten stocks (sparse portfolio).
- Sixth, to affirm the performance of the sparse portfolio, the thesis arranges a continuous dataset for training and testing during the whole 9-year dataset to obtain five different yearly performances.

The framework used in this thesis includes four phases: autoencoder phase, validation phase, calibration phase, and testing phase. The autoencoder phase aims to select desired stock components for the portfolio. The validation and calibration phases look for investment weights of stock components. The testing phase examines the out-of-sample performances of the beating portfolios.

The main research question of the thesis is *"Could the application of deep learning construct a sparse portfolio beating the S&P 500 Index?"*.

The rest of the thesis is structured as follows: chapter two presents an overview of the literature regarding the construction of the tracking portfolio with the sampling approach. Chapter three provides the detailed theoretical framework used to construct the methodology. Chapter four discusses the data and methodology. Chapter five presents the empirical results. The conclusion is given in Chapter six.

# 2 LITERATURE REVIEW

Analysis of historical performances of actively managed funds shows that the majority of them underperform the market in the long run (Rompotis, 2009). Thus, recently, passive investing has been discussed widely in the literature. As discussed, two main groups of methods used to track indices are synthetic replication and physical replication. In physical replication, two approaches are commonly used are full replication and sampling. Sampling seems to be the most promising approach amongst passive investing approaches in terms of transparency, flexibility, and transaction costs.

Two main problems that need to be solved when constructing the tracking portfolio with the sampling method are how to select a subset of constituents of the underlying index (asset selection) and how much to invest in each of them (asset weighing). Some researchers unify these two problems to solve them at once (joint approach), while others treat them as two separate problems and solve them in two steps (two-step approach).

In the joint approach, people usually turn the two problems into an optimization problem, which penalizes the cardinality of the tracking portfolio and provides the solutions of investment weights at once. In mathematical optimization terminology, the **objective function** is the function that we target to minimize or maximize. The objective function in the joint approach is usually the function of **tracking error** that is the difference between index's returns and portfolio's returns. Two sub-approaches under the joint approach are cardinality constrained optimization and regularized optimization.

In the two-step approach, we first select a subset of the assets and then allocate capital for the selected assets. Sub-approaches for asset selection can be divided into three groups: selection criteria, coverage of index structure, and optimized selection (Karlow, 2012). For asset weighting, two sub-approaches used are heuristic weighting and optimized weighting.

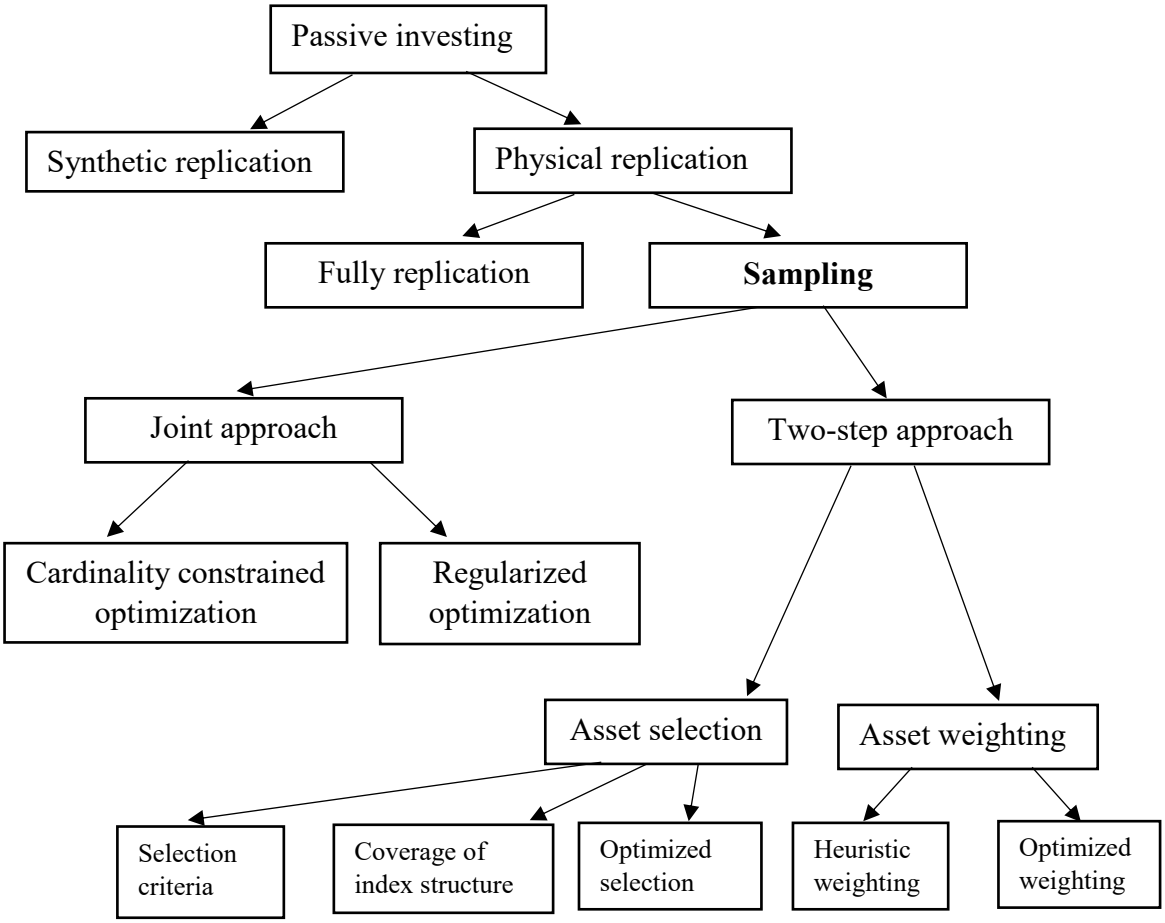


**Figure 1. Overview of passive investing methods**

## 2.1 Joint approach

### 2.1.1 Cardinality constrained optimization

Cardinality constraint is a constraint restricting the number of assets in the optimal portfolio. Some studies added a cardinality constraint that limits K assets included in the portfolio to objective functions. This is a mixed-integer nonlinear (quadratic) programming problem, which does not have a computationally effective solving algorithm (Chang, Meade & Beasley, 2000). Solving this problem can provide solutions for both asset selection and asset weighting. The cardinality constrained optimization was first applied in the mean-variance portfolio model, in which the objective function was the portfolio's variance (risk) function (Bienstock, 1996 & Chang et al., 2000). Beasley et al. (2003) and Ruiz-Torrubiano and Suarez (2009) then applied this cardinality constrained optimization in index tracking, in which the

objective function was the function of tracking error variance. Both groups of exact and heuristic approaches have been proposed in the literature to solve the complex mixed-integer nonlinear programming problem.

Bienstock (1996) followed the scholar of exact approaches presenting a branch-and-cut algorithm. However, Cesarone, Scozzari and Tardella (2010) claimed that Bienstock's method does not work effectively for the small number of restricted assets. Li, Sun and Wang (2006) introduced a convergent Lagrangian method to provide the exact optimal solution for the mean-variance portfolio problem. These exact methods can be generalized to apply in index tracking. However, Cesarone et al. (2013) claimed that exact approaches only partly solve the cardinality constrained portfolio optimization.

A lot of heuristic approaches have been proposed to solve the problem in index tracking, such as threshold accepting by Gilli and Kellezi (2002), genetic algorithms by Beasley et al. (2003), and Jeurissen and van den Berg (2008), simulated annealing by Derigs and Nickel (2004), and hybrid algorithms by Fastrich, Paterlini and Winker (2010), and Ruiz-Torrubiano and Suarez (2009), etc. However, the heuristic approach does not guarantee that the optimal solutions will be found. Still, the proper setup of the heuristic approach is able to find solutions close to the optimum (Karlow, 2012).

### 2.1.2 Regularized optimization

Mixed-integer nonlinear programming described in Section 2.1.1 is computationally heavy and impractical to put in use (Benidis et al., 2017). Thus, some studies proposed a more efficient algorithm to solve asset selection and asset allocation with the joint approach. Their studies applied $L_0$ regularization or $L_1$ regularization technique into the optimization problem to control the sparsity of the portfolio (Brodie et al. 2009, Wu et al., 2014 and Benidis et al., 2017). Specifically, the objective function in their study was constituted by $L_0$ (or $L_1$ regularization term) and the tracking error variance function. The method with $L_0$ regularization can be mathematically formulated as follows (the term $\lambda\|w\|_0$ below will be replaced by $\lambda\|w\|_1$ in $L_1$ regularization method):

$$w^* = argmin_w \frac{1}{m}\|R_I - R_x w\|_2^2 + \lambda\|w\|_0$$
s.t. $\sum_{i=1}^{n} w_i = 1,$
$w_i \geq 0$

where and $\lambda\|w\|_0$ is a regularization term (regularizer); $R_I \in \mathbb{R}^m$ is a vector of index returns in m periods; $R_x = [R_1, \ldots, R_n] \in \mathbb{R}^{mxn}$ is the return matrix of n component stocks in m periods; w = $[w_1, \ldots, w_n] \in \mathbb{R}^n$ is a vector of stock weights (so that $R_x w$ is the portfolio return).

While the constraint that all weights are positive restricts the short selling, the constraint that weights sum up to one represents the budget constraint. λ is the tuning hyperparameter which is priorly chosen and controls the sparsity of the portfolio. When λ goes to ∞, the impact of the regularizer gets larger, and the estimated weights of some assets approach zero. By enlarging the value of λ, we will get a sparser portfolio i.e. portfolio with a much smaller number of stocks. Besides the joint approach described above, Wu et al. (2014) also proposed the two-step approach that first selected stocks by applying $L_1$ regularization and then estimated the investment weight of each stock by using nonnegative least squares. They argued that their joint approach obtained poorer performance than their two-step approach.

However, Benidis et al. (2017) claimed that both $L_0$ regularization and $L_1$ regularization have their own problems in solving the asset selection problem. The objective function with $L_0$ regularizer is highly non-convex, which is not convenient for finding minima. $L_1$ regularization has a problem with the constraint that weights need to sum up to unity. This constraint reduces the $L_1$ regularizer $\lambda\|\mathrm{w}\|_1$ to a constant λ which is irrelevant to control the portfolio sparsity.

Takeda, Niranjan, Gotoh and Kawahara (2013)'s formulation included both the $L_2$ regularizer and the cardinality constraint. Unlike $L_0$ or $L_1$ regularization, $L_2$ regularization does not encourage the sparsity of the model. Alternatively, $L_2$ regularization can avoid large values of w, which can get rid of large variations. Thus, the $L_2$ regularizer in their formulation was responsible for enhancing the tracking portfolio's out-of-sample performance. Cardinality constraint was added to the

formulation to solve the stock selection problem. They claimed that their method could construct a portfolio containing a small number of stocks but still achieving good out-of-sample performance. Their formulation is given as:

$$w^* = argmin_w \frac{1}{m}\|R_I - R_x w\|_2^2 + \lambda\|w\|_2^2$$
$$\text{s.t. } \sum_{i=1}^{n} w_i = 1,$$
$$w_i \geq 0$$
$$\|w\|_0 \leq K$$

where K is a positive integer limiting the number of assets in the portfolio.

However, as they included cardinality constraint in their optimization problem, Takeda et al. (2013)'s method again faced the heavy computation problem of mixed-integer nonlinear programming as discussed in Section 2.1.1.

## 2.2 Two-step approach

### 2.2.1 Asset selection

#### 2.2.1.1 Selection criteria

In the selection criteria group of methods, a subset of assets is selected from the stock universe based on the set criteria.

One common criterion used is the weights of the assets in the index. As assets with larger weights in the index will contribute a larger impact on the variability of the index, assets that have weights satisfying with the set threshold will be selected. For example, if the tracking index is a capitalization-weighted index, the assets with the largest market capitalization will be included in the tracking portfolio (Meade & Salkin, 1989). When it comes to price-weighted indices, assets with smaller prices will have smaller weights and therefore be omitted (Alexander & Dimitriu, 2004a).

Rey and Seiler (2001) extended the asset weight criterion by integrating information of traded asset volumes into the market capitalization information. The assets that had a higher turnover-to-market capitalization ratio would be selected in their study. In fact, the research of Rey and Seiler showed that their combination method did not enhance the tracking portfolio's performance compared to the standalone capitalization-weighted criterion.

Rafaely and Bennell (2006) believed that the portfolio would closely track the market index if it could include the assets with larger contributions to the market index. They measured the contribution of an asset by calculating the product of the mean historic weight and the mean historic price of that asset. Assets with the larger product would have a larger contribution to the underlying assets and therefore be selected.

Montfort, Visser and van Draat (2008) used the correlation between the individual stock and stock universe as the selection criterion. They believed that assets with low correlation with market index are not beneficial for tracking index. In contrast, assets with a high correlation with the market index have a significant impact on the market index. However, Dunis and Ho (2005) claimed that correlation analysis on returns only reflects the short-run dependency between the asset components and market index. Thus, the tracking portfolios constructed with the correlation criterion need to be constantly rebalanced when the correlation varies. Such portfolio construction might be costly and difficult to manage. Dunis and Ho proposed the concept of cointegration as an alternative approach. They claimed that the cointegration reflects the long-term co-movements of the price series. Therefore, they constructed a portfolio cointegrated with the underlying index so that the tracking portfolio may not variate much from the index in the long run. Dunis and Ho believed that constructing the portfolio with cointegration analysis is not required to rebalance as frequently as one with correlation analysis.

Sorensen, Miller and Ooi (2000) used the decision tree technique to build more complex selecting criteria, including valuation, profitability, earnings, etc. They used classification and regression tree (CART) analysis to detect which assets are likely to outperform the index based on different criteria. A decision tree model used six variables representing the criteria such as sales-to-price ratio, cash flow-to-price ratio,

return on assets, etc. to provide the probabilities of stock's outperformance and stock's underperformance. The approach provided the framework to classify the performance of stocks. Furthermore, the approach determined which criteria were important for the performance classification.

#### 2.2.1.2 Optimized selection

The methods under this group pick stocks by forming and solving an optimization problem.

Avoiding the complexity of the mixed-integer nonlinear programming as discussed in Section 2.1.1, Gaivoronski, Krylov and Van Der Wijst (2005) established a simple method for stock selection. They first formulated the minimization problem with the objective function of tracking error. They provided the weight solution for all the constituents of the benchmark index by solving the optimization problem. After that, assets with the largest weights were picked for the tracking portfolio. However, as some stocks were highly correlated, the optimization problem formulated by Gaivoronski et al. (2005) could allocate small weights to a whole group of those stocks. Therefore, Montfort et al. (2008) claimed that this approach could lead to the omission of the whole segment of stocks that were highly correlated with each other from the tracking portfolio. Coleman, Li and Henniger (2006) suggested the reverse sequence for the method. Instead of choosing stocks with the largest weights, they proposed to exclude a very small number of stocks with the smallest weights, then repeating the optimization problem for the rest until getting the desired number of stocks in the tracking portfolio.

The strategy of selecting stocks with the largest weights was also implemented in Wu et al. (2014)'s study. They added the regularizer into the tracking error variance function to create the objective function, as discussed in Section 2.1.2. They implemented both the joint approach and the two-step approach. In the two-step approach, they first selected stocks by applying $L_1$ regularization and then estimated the investment weight of each stock by using nonnegative least squares. When comparing with their joint approach, they claimed that the two-step approach obtained

a better performance. However, using $L_0$ or $L_1$ regularization for stock selection is not thoroughly applicable in index tracking, as discussed in Section 2.1.2.

#### 2.2.1.3 Coverage of index structure

The methods belong to this group attempt to construct a portfolio containing only assets representing the index feature. Specifically, the methods will remove assets that have the same information from the tracking portfolio. Thus, a portfolio containing a subset of assets can still mostly cover the index structure.

Maginn, Tuttle, McLeavey and Pinto (2007, p. 425) used stratified sampling to mimic the structure of the index. Based on the pre-defined features of the index, the method separated the stock universe into different segments representing the index features. For example, an index feature could be large market capitalization, medium market capitalization, or small market capitalization; then, stocks would be categorized according to such index feature. It is also possible to combine two or more features together. For example, the feature of industry sectors could be combined with market capitalization. Assets were then categorized in different groups of combined features, such as the group of large-market-capitalization assets in the technology sector and the group of small-market-capitalization assets in the health care sector, etc. The study then selected the most representative asset of each group for the tracking portfolio.

Focardi and Fabozzi (2014) argued that the hierarchical clustering could reveal the correlation and cointegration between stock constituents. They first used historic price time series of stocks to calculate the Euclidean distance between clusters. Euclidean distance in their study was the minimum price distance between all pairs of stock elements. Stock components were then grouped in the different clusters based on the Euclidean distance. After determining the optimal number of clusters, a tracking portfolio was constructed by selecting one stock or a subset of stocks from each cluster and calculating each asset's weight. Focardi and Fabozzi introduced three basic strategies. The first was a semi-automatic strategy that partly gave the stock selection task to managers. The managers selected the stocks from clusters based on their experience and judgment. The optimizer then calculated the weights of the selected assets. The second was an automatic strategy that used the heuristic approach to pick

stocks e.g. picking stock based on the maximum return criterion. Then the optimizer solved the asset weighting problem. The third was a fully quantitative strategy, in which the optimizer solved both asset selection and asset weighting problems. Thus, the third strategy requires heavy computation, especially in the case of selecting a subset of stocks from each cluster.

Corielli and Marcellino (2006) used principal component analysis to extract needful features from the input data. Principal component analysis extracts the input features by reducing the input dimensionality. They first built a linear factor model for the index, in which index prices were explained by several factors and an error term. The linear factor model was determined by principal component analysis on the matrix of stock prices. The number of factors was smaller than the dimensionality of the input. Stocks correlated with the index factors were selected for the tracking portfolio so that the tracking portfolio shared the same factor structure with the index. Alexander and Dimitriu (2004b) also used principal component analysis to construct a tracking portfolio. In their principal component analysis, the first principal component was the linear combination between the input variables with maximum variation. Alexander and Dimitriu claimed that the first principal component could capture the maximum variation on the stock returns; they, therefore, constructed a tracking portfolio replicating the first principal component instead of the market index.

Another method used for dimensionality reduction is autoencoder. Autoencoder is unsupervised learning using an artificial neural network to extract the input features and recreate the input itself based on the extract features (Goodfellow, Bengio & and Courville 2016, p. 502). Autoencoder is processed in three steps: encoder, bottleneck and decoder. In the encoding step, input is multiplied with an appropriate weight value and added a bias term. The result of that calculation goes through an activation function in the bottleneck layer to discover its latent state presentation. Finally, the decoding step reproduces the input from the latent state presentation. The process of autoencoder attempts to minimize the difference between the original input and decoded input which is called the reconstruction error. Heaton et al. (2017b) used an autoencoder for stock selection. They combined stocks with the smallest reconstruction error and the largest reconstruction error to form the tracking portfolio. They claimed that this portfolio construction could avoid selecting stocks containing

the same information. After constructing the portfolio, they used the artificial neural network to map selected stocks' returns with the index returns. Their empirical results showed that their tracking portfolio could beat the Biotechnology IBB Index by 1% annually.

### 2.2.2 Asset weighting

#### 2.2.2.1 Heuristic weighting

Heuristic weighting is a weighting scheme built on simple and, arguably, rational rules. The approach may allocate the assets in the portfolio based on the weighting methodologies of the benchmark indices. For example, if the benchmark index is S&P 500, a capitalization-weighted index, then the heuristic-weighting method allocates the portfolio assets based on their capitalization in the market (Larsen & Resnick, 1998). Thus, we will have four heuristic weighting methods corresponding to four different index weighting methodologies: capitalization weighting, price weighting, equal weighting, and fundamental weighting.

The heuristic weighting approach is simple to apply; however, the approach may be too naïve because it overlooks the correlation of assets in the index structure. Thus, applying this approach in sampling may not bring the optimal results in index tracking.

#### 2.2.2.2 Optimized weighting

Wu et al. (2014) used the linear regression fitting the portfolio returns to index returns with the constraints on weights (short-selling restriction and weights summing up to unity) to determine assets' weights. This method is called a non-negative least square. The least-square method fits the linear regression by minimizing the difference between the input and the output variables i.e. minimizing tracking error variance. The non-negative least square is given as:

$$w^* = argmin_w \|R_I - R_x w\|_2^2$$
s.t. $\sum_{i=1}^{n} w_i = 1,$
$w_i \geq 0$

where $R_I \in \mathbb{R}^m$ is a vector of index returns in m periods; $R_x = [R_1, \ldots, R_n] \in \mathbb{R}^{mxn}$ is the return matrix of n component stocks in m periods; w = $[w_1, \ldots, w_n] \in \mathbb{R}^n$ is a vector of stock weights (so that $R_x w$ is the portfolio return).

Tracking error and tracking error variance are the most common objective functions in index tracking. However, Gaivoronski et al. (2005) and Rafaely and Bennell (2006) used a less common tracking quality function, which is the difference between portfolio values and index values, to determine investment weights.

The type of optimization problem varies depending on which tracking quality function is used. It can be a quadratic optimization problem if the objective function is tracking error variance or mean squared error (Roll, 1992 and Beasley et al., 2003). Or it can be a linear optimization problem if the used measures are linear, such as mean absolute deviation, maximal absolute deviation, etc. (Rudolf, Wolter & Zimmermann, 1999).

# 3 THEORETICAL FRAMEWORK

Since autoencoder is the type of deep neural network, Sections 3.1 and 3.2 provide the theoretical framework of deep learning and autoencoder, which is used to construct the methodology for solving the stock selection problem. The study discusses the similarity between the traditional Capital Asset Pricing Model (CAPM) and the deep autoencoder model in Section 3.3. Section 3.4 introduces the regularization technique used to construct the stock weighting methodology.

Deep learning or deep neural network is a subset of machine learning. Like machine learning, deep learning is also the learning algorithm that can learn from the experience E when performing the task T to improve the performance measured by P (Mitchell 1997, p.99).

a) Task T in deep learning

There are different types of tasks solved by deep learning. The most common tasks in deep learning are classification and regression problems. Performing the tasks in deep learning is illustrated by how deep learning would process an example. The **example** mentioned here means the vector $x = [x_1, x_2,\ldots, x_n]$, where $x_1, x_2,\ldots, x_n$ are n features of the input. For example, if the input is the returns of ten different stocks in m periods, ten stocks are called ten features, and the returns of ten stocks in one period is called an example.

In the classification task, deep learning is asked to categorize the class of an example. Binary classification and multi-class classification are two typical problems in the classification task. Binary classification asks to classify an example to one of two classes (e.g. classify whether an email is a spam or not). Multi-class classification has more than two classes in its classification problem (e.g. classify whether an image is an orange, an apple or a peach).

In the regression task, the problem that needs to be solved is to predict real-number values when the input is given. One of the applications of this type of prediction is algorithmic trading. For example, deep learning can perform the regression task of

predicting stock prices when macroeconomic variables and firm-specific characteristics are given as the input.

b) Measure P in deep learning

The choice of performance measure P is associated with the task T. For the classification problem, the performance measure P is usually the accuracy of the models. The performance measure of accuracy is the percentage of predicted correct examples over the total predicted examples. In the regression task, the performance measure P is the error, which is the difference between the predicted values and the true values.

c) Experience E in deep learning

Based on the experience that learning algorithms obtain during the learning process, learning algorithms can be categorized into supervised learning algorithms and unsupervised learning algorithms.

**Supervised learning** experiences the dataset containing many input features and a label y associated with each example. The supervised learning experiences the relationship between the input and the label y, so that the algorithms can predict output y when an example is given. In other words, supervised learning attempts to estimate the conditional probability distribution $p(y \mid x)$. For example, using deep neural networks to learn the relationship between index returns (label y) and stock components' returns (feature x) is supervised learning.

In contrast, there is no label y given in the dataset in unsupervised learning. Unsupervised learning does not experience the relationship between the input x and the label y. **Unsupervised learning** experiences the dataset including many input features to discover the dataset structure. Or, put differently, unsupervised learning aims to learn the entire probability distribution of the dataset i.e. the joint probability distribution of all observed data points $p(x)$. For example, in deep learning, autoencoder is an unsupervised learning method. Autoencoder can explore the structure of a given dataset by extracting the attributes of the dataset structure.

Therefore, autoencoder seems to be a promising method for selecting a subset of stocks mimicking the index market structure.

From the above, when the thesis claims that it would like to use autoencoder for solving stock selection problem, there are two main points to be noted:

- First, the stock selection is the regression task.
- Second, the autoencoder is the unsupervised learning method.

## 3.1 Deep neural network

Linear models limit the relationship between the input and the output to a linear form and omit the interaction between input variables. Deep feedforward networks (or feedforward neural networks or deep neural networks) attempt to enhance the linear model performance by overcoming the linear model's limitations. Feedforward neural networks can explore the nonlinear relationship between the input and the output by nonlinearly transforming the input and mapping the transformed input with the output. In other words, input x is transformed through $\varphi(x)$, where $\varphi$ is a nonlinear transformation, then this x's new presentation $\varphi(x)$ is mapped with the output y. We need to determine the parameters doing this mapping task to build a deep neural network model.

The function $\varphi$ in neural networks is not necessarily a nonlinear transformation; it can be a linear transformation. If a neural network only consists of linear transformation functions, such neural network turns into the linear model. The function $\varphi$ is called the **activation function**. The activation function can appear in both the hidden layer and the output layer in neural networks.

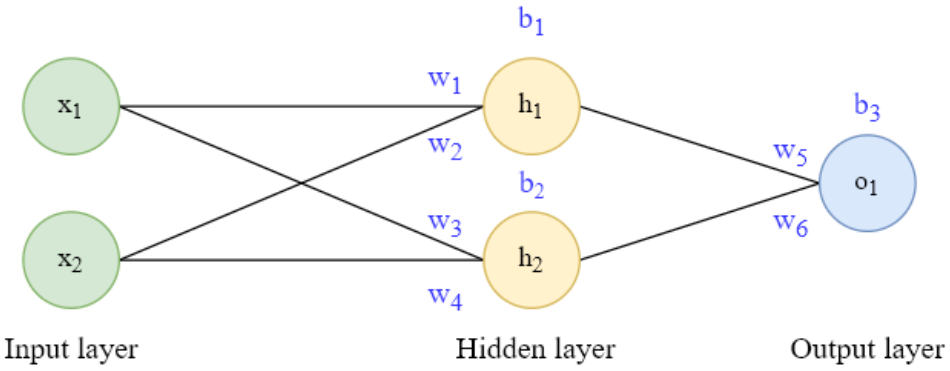


**Figure 2. Simple feedforward neural network**

The feedforward neural networks have "feedforward" in their name as the information flows from the input layer, then is transformed in the hidden layer, and finally, the transformed information in the hidden layer is used to produce the output. The nodes that data and computations flow through are called **neurons.**

Figure 2 shows the simple feedforward neural network, including an input layer, a hidden layer, and an output layer. This network has two inputs ($x_1$ and $x_2$) in its input layer; two neurons ($h_1$ and $h_2$ nodes) in its hidden layer; and one output in its output layer. The mapping procedure between the input and the output in this simple feedforward neural network is described below.

First, the inputs are multiplied by the **weight w**; then such multiplications are summed together and then are added the **bias b**, as details:

$$(x_1 * w_1) + (x_2 * w_2) + b_1 \quad (3.1)$$

$$(x_1 * w_3) + (x_2 * w_4) + b_2 \quad (3.2)$$

Then the sums will be gone through the hidden layer, in which the activation function transforms the sums into $h_1$ and $h_2$:

$$h_1 = f_1(x_1 * w_1 + x_2 * w_2 + b_1) \quad (3.3)$$

$$h_2 = f_1(x_1 * w_3 + x_2 * w_4 + b_2) \quad (3.4)$$

Then $h_1$ and $h_2$ are again multiplied by the weight w. Such multiplications are summed together and then are added the bias b, given as:

$$h_1 * w_5 + h_2 * w_6 + b_3 \tag{3.5}$$

Then the sum is transformed through the activation function in the output layer to produce the output:

$$o_1 = f_2(h_1 * w_5 + h_2 * w_6 + b_3) \tag{3.6}$$

The above illustration is the simple version of the feedforward neural network consisting of only one hidden layer. The feedforward neural network can have multiple hidden layers and those layers can consist of any number of neurons. The greater number of hidden layers, the deeper the models. The "deep learning" terminology comes from the neural network's depth represented by the number of hidden layers. Figure 3 illustrates the deeper feedforward neural network with multiple hidden layers, multiple inputs and multiple outputs.

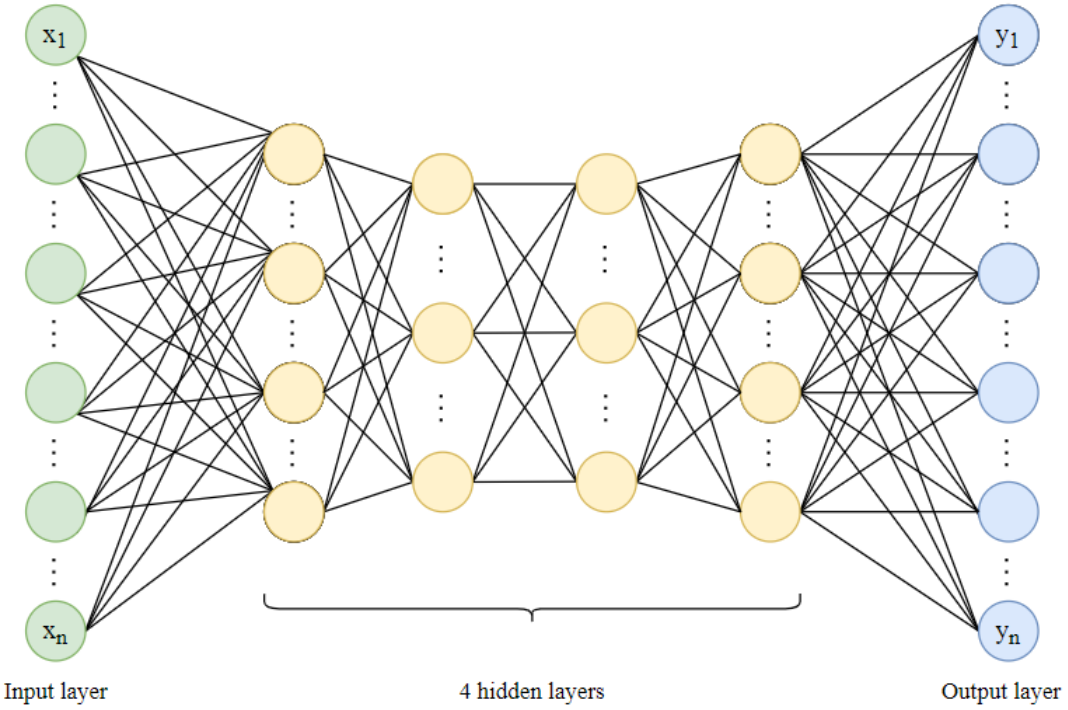


**Figure 3. Deep feedforward neural network**

Below is a summary of the feedforward neural network process.

When the input X and output Y are given, the feedforward neural network attempts to find the parameters to map the input to the output through the layers. Let $Z_{(l)}$ denote the information generated from the $l^{th}$ layer of the neural network i.e. $Z_{(l)}$ are extracted features from the $l^{th}$ layer, so $X = Z_{(0)}$. Then the framework of the feedforward neural network model is as follows:

$$
\begin{aligned}
&Z_1 = f_1(W_1 X + b_1),\\
&Z_2 = f_2(W_2 Z_1 + b_2),\\
&\ldots.\\
&Z_{L-1} = f_{L-1}(W_{L-1} Z_{L-2} + b_{L-1}),\\
&\hat{Y} = f_L(W_L Z_{L-1} + b_L).
\end{aligned} \tag{3.7}
$$

where W is weight matrices and b is the bias vector.

In short:

$$\hat{Y} = f_L(W_L f_{L-1}(\ldots W_2 f_1(W_1 X + b_1) + b_2 \ldots) + b_L). \tag{3.8}$$

The training problem of the feedforward neural network models is to find the model parameters $\widehat{W} = (W_1, \ldots, W_L)$ and $\hat{b} = (b_1, \ldots, b_L)$. The model parameters are determined by minimizing the cost function that measures the difference between true values and predicted values. The next section introduces the concept of the cost function and how to choose the appropriate cost function in neural networks.

### 3.1.1 Cost function

The target function solved in minimization or maximization problems is called the objective function. The goal in any predicting model is to minimize the error i.e. the difference between the true values and the predicted values. The objective function measuring the error of the models is called the **cost function** or loss function. The two main types of a cost function using in training neural network models are mean squared

error and cross-entropy. The choice of cost function should suit the tasks that deep learning attempts to perform. As discussed earlier, the main tasks in deep learning are to solve regression and classification problems.

3.1.1.1 Mean squared error

The thesis first discusses Euclidean distance as it closely connects with mean squared error. Euclidean distance is the most popular distance metric in machine learning. Euclidean distance is used to measure the distance between the points in an n-dimensional space.

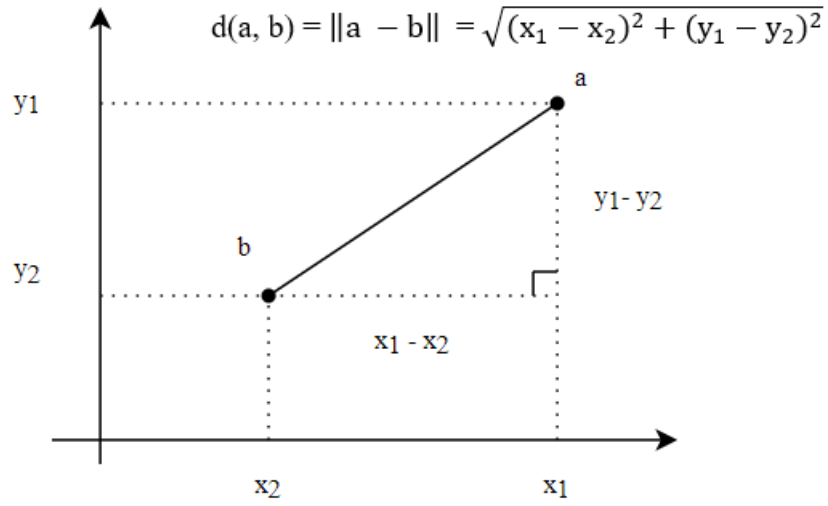


**Figure 4. Euclidean distance between two points in two-dimensional space**

The Euclidean distance between point a and point b in the space is often denoted as $\|a - b\|_2$ or simply as $\|a - b\|$.

In the case of two-dimensional space described in Figure 4, the Euclidean distance between point a and point b denoted as d(a, b) is given as:

$$d(a, b) = \|a - b\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3.9)$$

The Euclidean distance can be generalized in a N-dimensional space, where point A is represented as ($x_1$, $x_2$, ..., $x_N$) and point B is represented as ($y_1$, $y_2$, ..., $y_N$). Then, the Euclidean distance between point a and point b is given as:

$$d(a, b) = \|a - b\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2} \quad (3.10)$$

A cost function can be defined as Euclidean distance measuring the distance between the two vectors: vector of true values and vector of predicted values; then we call the cost function as least squared error or **$L_2$ loss** ($L_2$ norm). Thus, $L_2$ loss is formulated as the square root of the sum of squared errors. The formula of $L_2$ loss is described as follows:

$$L^2 = \|y^{true} - y^{predict}\| = \sqrt{\sum_{i=1}^{m}(y_i^{true} - y_i^{pred})^2} \quad (3.11)$$

where $y_i^{pred}$ and $y_i^{true}$ are the predicted value and the true value of the output, respectively when the $i^{th}$ example is given; m is the number of the training examples.

**Mean squared error** (MSE) is defined as the average of the sum of squared errors. The MSE is described in the formula as follows:

$$\text{MSE} = \frac{1}{m}\sum_{i}^{m}(y_i^{true} - y_i^{predict})^2 = \frac{1}{m}\|y^{true} - y^{predict}\|^2 \quad (3.12)$$

The formula of MSE (3.12) shows that MSE is the mean squared $L_2$ loss. $L_2$ loss and MSE are basically the same concept which measures the distance between the predicted value and the true value. Both $L_2$ loss and MSE are used as the cost function in machine learning and deep learning. Some people use the terms $L_2$ loss and MSE interchangeably.

As the regression problem is associated with predicting a real-valued output, MSE/ $L_2$ loss is the most common cost function used for a regression problem. However, MSE/ $L_2$ loss is badly defined in a classification problem as the classification predictive model attempts to predict the discrete output variables. Alternatively, cross-entropy is the common cost function used in a classification problem.

3.1.1.2 Maximum likelihood estimation

Before diving into the cross-entropy concept, the thesis first discusses maximum likelihood estimation as it closely connects with the concept of cross-entropy. Maximum likelihood estimation (MLE) is a statistical technique using some observed data to estimate the parameters of a given probability distribution.

For unsupervised learning, considering a set of m examples of observation $X = \{x^{(1)}, \ldots, x^{(m)}\}$, where each example is drawn independently from the same but unknown probability distribution $p_{data}(x)$ (so-called independent and identically distributed assumption i.i.d.). MLE attempts to estimate the parameters of a probability distribution function $p_{model}(x;\theta)$ which best explains $p_{data}(x)$, where θ is the parameters of a probability distribution function. For example, in the normal distribution i.e. Gaussian distribution, θ will represent two parameters: the mean and the standard deviation.

When the assumption of i.i.d. is made, the total probability of all observed data is the product of the probability of each data point individually. The maximum likelihood estimator for θ is then described as:

$$\theta_{ML} = \underset{\theta}{argmax}\, p_{model}(X;\theta) \tag{3.13}$$

$$= \underset{\theta}{argmax} \prod_{i=1}^{m} p_{model}\left(x^{(i)};\theta\right) \tag{3.14}$$

However, as the product of many probabilities can be inconvenient for finding optimum, the maximum likelihood estimator is transformed to the logarithm form which does not affect its argmax value. By doing this, the product is transformed into a sum:

$$\theta_{ML} = \underset{\theta}{argmax} \sum_{i=1}^{m} log\, p_{model}\left(x^{(i)};\theta\right) \tag{3.15}$$

The objective function in (3.15) can be divided by m to get another version of maximum likelihood estimator without changing the argmax value. This version can be expressed as an expectation to the distribution of the training data $\hat{p}_{data}$:

$$\theta_{ML} = \underset{\theta}{argmax}\, \mathrm{E}_{x\sim\hat{p}data}\, log\, p_{model}(x;\theta) \quad (3.16)$$

3.1.1.3 Cross-entropy

The above section interprets the MLE as a technique estimating the parameters of a probability distribution function $p_{model}(x;\theta)$ which best explains $p_{data}(x)$. However, there is another interpretation of MLE that can unravel the concept of cross-entropy.

MLE can be interpreted as the minimization of the dissimilarity between the distribution of the training data ($\hat{p}_{data}$) and the model distribution ($p_{model}(x;\theta)$). The degree of the dissimilarity is given as:

$$\text{Minimize} \quad \mathrm{E}_{x\sim\hat{p}data}[\, log\, \hat{p}_{data}(x) - log\, p_{model}(x)] \quad (3.17)$$

$log\, \hat{p}_{data}(x)$ is the process of generating the training data that cannot be changed. Thus, to minimize the objective function in (3.17), we only need to minimize the negative log-likelihood:

$$\text{Minimize} \quad -\mathrm{E}_{x\sim\hat{p}data}[\, log\, p_{model}(x)] \quad (3.18)$$

which is the same as the maximization in equation (3.16).

The negative log-likelihood in (3.18) is called the **cross-entropy** between the probability distribution of the training data $\hat{p}_{data}$ and the probability distribution estimated by the model $p_{model}(x;\theta)$. Thus, MLE is equivalent to minimizing the negative log-likelihood and equivalent to minimizing the cross-entropy.

3.1.1.4 Relation of MSE and cross-entropy

If the probability distribution $p_{data}(x)$ is assumed to follow a Gaussian (normal distribution), then minimizing cross-entropy becomes:

$$\text{Minimize} - \mathrm{E}_{x\sim\hat{p}data}[\ log\ p_{model}(x)] \tag{3.19}$$

$$\approx \quad \text{Minimize} \quad - \mathrm{E}_{x\sim\hat{p}data}\left[\ log\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x^{true}-x^{test})^2}{2\sigma^2}}\right]$$

$$\approx\ Minimize\ -\mathrm{E}_{x\sim\hat{p}data}\left[log\ (\frac{1}{\sqrt{2\pi\sigma^2}})\ - \frac{(x^{true}\ -\ \mu)^2}{2\sigma^2}\right]$$

$$\approx\ Minimize\ -\mathrm{E}_{x\sim\hat{p}data}\left[log\ (\frac{1}{\sqrt{2\pi\sigma^2}})\ - \frac{(x^{true}-x^{test})^2}{2\sigma^2}\right]$$

$$\approx\ Minimize\ \mathrm{E}_{x\sim\hat{p}data}(x^{true}\ - x^{test})^2 \tag{3.20}$$

where $\mu$ and $\sigma$ are the estimated mean and the standard deviation of the probability distribution $p_{data}(x)$, respectively.

Recalling the MSE formula (3.12), we can see that when the target distribution is assumed to follow a Gaussian, minimizing either MSE or cross-entropy leads to the same optimum.

There are several points to be noted under this section:

- First, MLE is equivalent to minimizing the negative log-likelihood and equivalent to minimizing the cross-entropy.
- Second, using either MSE or cross-entropy as a cost function leads to the same optimum when the Gaussian distribution assumption is made. Thus, in the regression problem, we can use either MSE or cross-entropy as a cost function.

After this section, we can understand the concept of cost function and how to choose an appropriate cost function in the regression problem. The next sections will introduce the techniques to solve the cost function: stochastic gradient descent and back-propagation algorithms.

### 3.1.2 Gradient-Based Learning

#### 3.1.2.1 Gradient Descent

Given a cost function $J(\theta) = \frac{1}{m}\sum_i^m (y(\theta)_i^{predict}\ -\ y_i^{true})^2$, where $\theta$ are parameters of the deep neural network model including the weights w and the biases b, $J'(\theta)$ provides

the slope (or gradient) of $J(\theta)$ at the point $\theta$. In other words, $J'(\theta)$ is the derivative of function $J$ shows how $J$ changes when there is a unit change in $\theta$ at a specific point $\theta$. Thus:

$$J(\theta + \epsilon) \approx J(\theta) + \epsilon J'(\theta) \tag{3.21}$$

where $\epsilon$ can be understood as the degree of the change of $\theta$ and $J'(\theta)$ is the derivative of $J(\theta)$. Another notation for $J'(\theta)$ is $\frac{dJ(\theta)}{d\theta}$.

We can adjust the movement direction of $\theta$ to reduce or increase the value of $J$ based on the sign of slope i.e. the sign of the function's derivative $J'(\theta)$. Thus, the derivative of the objective function is widely used in solving the optimization problem. For the purpose of convenient illustration, let simplify the cost function $J(\theta) = \frac{1}{2}\theta^2$ representing the convex function. The detailed illustration is described in Figure 5; we can see that $J'(\theta) < 0$ when $\theta < 0$, so $J(\theta)$ will decrease when we move $\theta$ rightward. In contrast, $J'(\theta) > 0$ when $\theta > 0$, $J(\theta)$ will decrease when we move $\theta$ leftward. This example illustrates that the sign of $J(\theta)$'s derivative helps determine the movement direction of $\theta$ to decrease $J$. Specifically, $J$ is reduced when we move $\theta$ in small steps with the sign opposite with its derivative:

$$J(\theta - \epsilon * sign(J'(\theta)) < J(\theta) \text{ for small enough } \epsilon \tag{3.22}$$

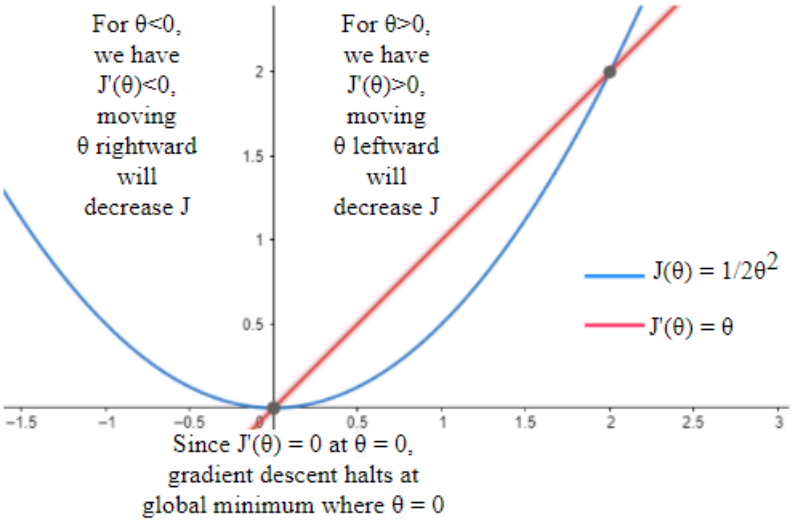


**Figure 5. An illustration of gradient descent (1) (adapted from Goodfellow, Bengio & Courville 2016, p. 83)**

As $\theta$ includes multiples variables of weights w and biases b. The gradient descent technique can be generalized in the case of multiple $\theta$ by using the concept of partial derivatives. The partial derivative is denoted as $\frac{\partial J(\theta)}{\partial \theta j}$ measuring the change in $J(\theta)$ corresponding to a unit change of $\theta_j$ while other values of $\theta$ keep unchanged. The gradient of $J(\theta)$ with multiple values of $\theta$ is a vector containing all the partial derivatives with respect to $\theta_j$. **The gradient vector** is given as**:**

$$\nabla_\theta J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_1}, \ldots, \frac{\partial J(\theta)}{\partial \theta_L}\right) \tag{3.23}$$

where L is the number of layers in the model.

As is the case for cost function with single variable $\theta$ discussed above, we can decrease function J($\theta$) in the case of multiple $\theta$ by moving $\theta$ in small steps in the opposite direction with the gradient. This technique is called **steepest descent** or **gradient descent**. The new point $\theta'$ after the movement is:

$$\theta' = \theta - \epsilon \nabla_\theta J(\theta) \tag{3.24}$$

where $\epsilon$ is the positive scalar value representing the size of the step; in machine learning terminology, $\epsilon$ is called the **learning rate**. The value of $J$ will descend gradually for each iteration of the learning step described in Figure 6. The learning rate $\epsilon$ is a hyperparameter meaning that it must be priorly chosen by the operator of the machine learning algorithm. The learning rate is chosen based on the features of the problem and given data.

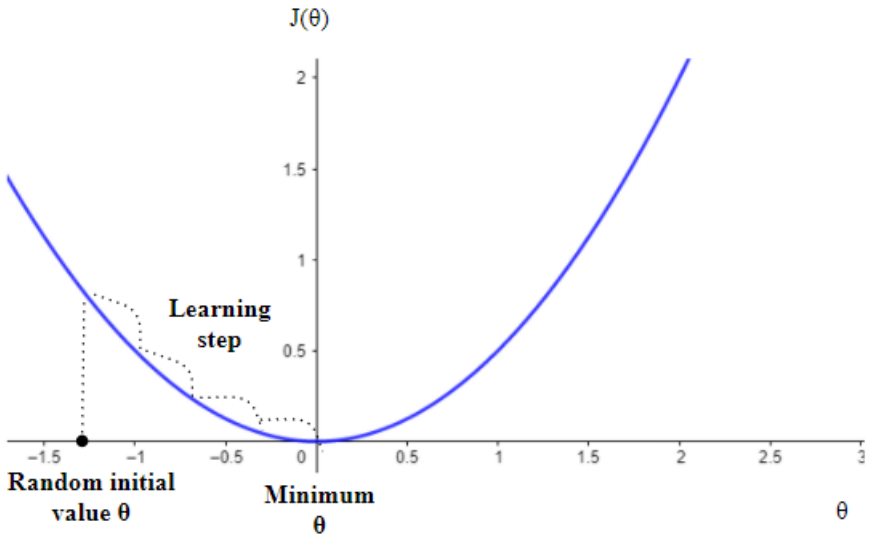


**Figure 6. An illustration of gradient descent (2)**

If the derivative $J'(\theta)$ at the point $\theta$ equals 0, then the derivative at such point gives no indication for the movement direction of $\theta$. Such point is called a **critical point** or a **stationary point**. In the problem with multidimensional $\theta$, the critical point of the cost function is the point where all elements of the gradient vector at such point are equal to zero. The critical point can be a local maximum, a local minimum or a saddle point. We say a function J has a **local minimum** at the point $\theta$ when the values $J$ at all the neighboring points of $\theta$ is larger than $J(\theta)$. In contrast, a function $J$ has a **local maximum** at the point $\theta$ when the values of $J$ at all the neighboring points of $\theta$ is smaller than $J(\theta)$. A **saddle point** is a critical point but it is neither a maximum nor a minimum, meaning that its neighbors are both lower and higher than itself. The local minimum and the saddle point are illustrated in Figure 7.

A point where function $J$ obtains the smallest value is a **global minimum**. The most desired outcome when minimizing the cost function in deep learning is to find a global

minimum. However, it is difficult for optimization algorithms in deep learning to determine the global minimum when the cost function has many local minima or saddle points. In such scenario, we generally accept the found point that is significantly low enough. This illustration is described in Figure 7.

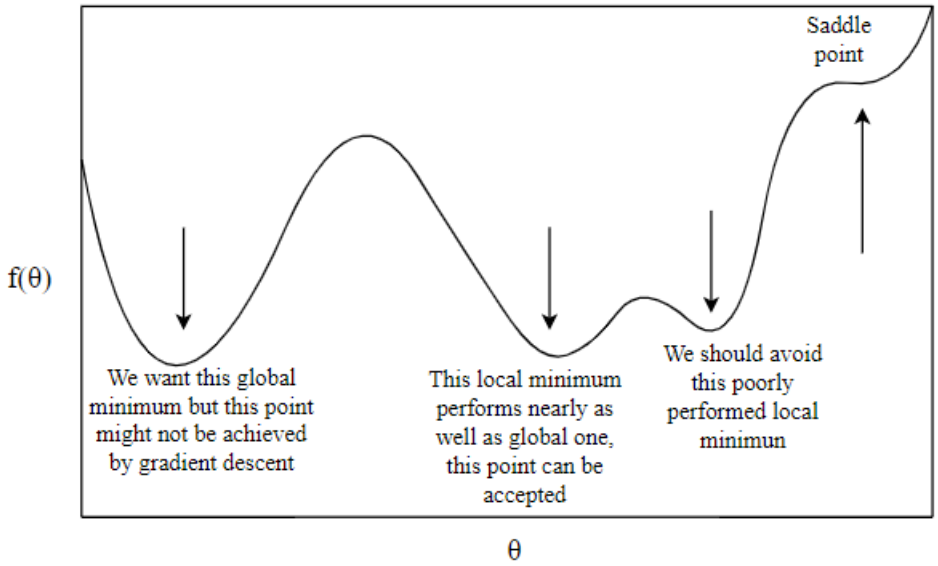


**Figure 7. An illustration of critical points (adapted from Goodfellow, Bengio & Courville 2016, p. 85)**

3.1.2.2 Stochastic Gradient Descent

In fact, when the training set is huge containing millions of examples, then the use of gradient descent causes the expensive computation. Gradient descent must run through all the examples in the training set for each update of the parameter $\theta$ in a particular iteration. Each iteration when using gradient descent to train a regression model is given as:

$$J_{train}(\theta) = \frac{1}{m}\sum_i^m (y_i^{predict} - y_i^{true})^2, \quad (3.25)$$

$$y_i^{predict} = \sum_{j=0}^{L} \theta_j x_j \quad (3.26)$$

Repeat until convergence {

$$\theta_j := \theta_j - \epsilon \frac{2}{m}\sum_i^m (y_i^{predict} - y_i^{true}) x_{ij} \quad (3.27)$$

(for every $j = 0, \ldots, L$)

}

where m is the number of examples and L is the number of layers.

Stochastic gradient descent (SGD) is the extension of gradient descent to solve the computation problem caused by the huge training set. SGD is essential for training models in deep learning. In SGD, we need to run only a subset of examples for each update of the parameter in a particular iteration. This helps to reduce the heavy computation when using gradient descent in large training sets. A subset of examples is called **minibatch** in SGD, which is a relatively small number of examples (m') compared to the huge number of examples (m) in training set; m' is chosen by the operator. Generally, a minibatch contains from one to a few hundred examples B = $\{x^{(1)}, \ldots, x^{(m')}\}$. In the regression model, the SGD algorithm for the minibatch of one example i.e. $m' = 1$ is formed as:

$$J_{train}(\theta) = \frac{1}{m}\sum_i^m (y_i^{predict} - y_i^{true})^2, \tag{3.28}$$

$$y_i^{predict} = \sum_{J=0}^{L} \theta_j x_j \tag{3.29}$$

Repeat until convergence {

for i := $1, \ldots, m$ {

$$\theta_j := \theta_j - 2\epsilon(y_i^{predict} - y_i^{true})x_{ij} \tag{3.30}$$

(for every $j = 0, \ldots, L$)

}

}

As we know that the neural network model requires that the data be forwardly transformed through multiple layers. Thus, it is required to go backward through each layer to take the derivative of the cost function; this technique is called Back-Propagation.

### 3.1.3 Back-Propagation

We have discussed the stochastic gradient descent algorithm finding the minimum of a cost function. The main idea of the gradient descent technique is to look for the partial derivative with respect to each model parameter $\frac{\partial J(\theta)}{\partial \theta j}$. Then we will base on the

sign of its partial derivative to determine which direction of the particular parameter should move to reach the local minimum i.e. moving the parameter in multiple small steps with the sign opposite with its partial derivative until convergence.

As the data is transformed through different activation functions in different layers in the deep neural network, the partial derivative $\frac{\partial J(\theta)}{\partial \theta j}$ cannot be obtained straightforwardly. The technique used to take the partial derivative $\frac{\partial J(\theta)}{\partial \theta j}$ in deep learning is the backpropagation algorithm.

The **backpropagation algorithm** uses the chain rule method to compute the cost function's gradient with respect to each parameter $\frac{\partial J(\theta)}{\partial \theta j}$. The chain rule can be briefly explained as follows: if a variable z depends on a variable y, and the variable y depends on another variable x, then z, through the variable y, also depends on x i.e. if z = f(y) and y = g(x), so that z = f(y) = f(g(x)). The chain rule then states that:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{3.31}$$

The thesis will go back with the example of the simple neural network at the beginning of Section 3.1 to illustrate how the backpropagation algorithm uses the chain rule method to find a local minimum of a cost function.

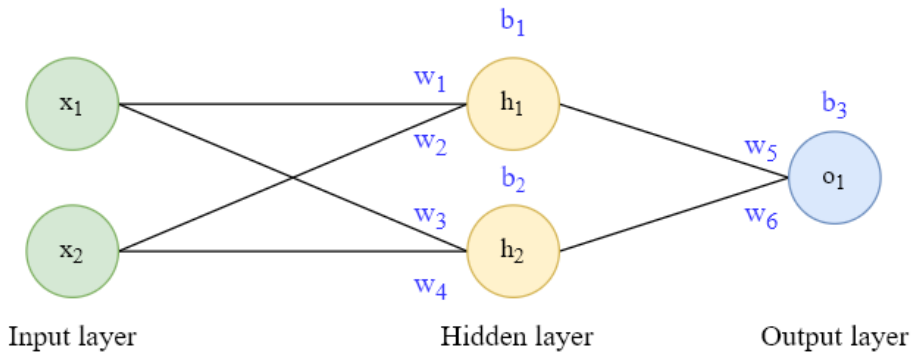


**Figure 8. Simple feedforward neural network**

To simplify the case, let all the activation functions in the output layer and the hidden layer $o_1$, $h_1$ and $h_2$ be all linear activations i.e. no transformation through the activation function. Let the problem in the example in Section 3.1 be a regression problem; then we know that the used cost function is MSE as discussed in Section 3.1.1. For the purpose of illustration, the data is given with only one example as:

**Table 1. Example data**

| No | X1 | X2 | Y |
|---|---|---|---|
| 1 | 2 | 3 | 1 |

To use gradient descent to find the local minimum, we first need to get the partial derivative $\frac{\partial J}{\partial w_1}$. Since $w_1$ only affects $h_1$ (not $h_2$), we can apply the chain rule for $\frac{\partial J}{\partial w_1}$ as:

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial y^{pred}} * \frac{\partial y^{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

The MSE for the single example is given as:

$$J_{train}(\theta) = \frac{1}{1}\sum_1^1 (y^{true} - y^{pred})^2$$
$$= (y^{true} - y^{pred})^2$$
$$= (1 - y^{pred})^2$$

Thus:

$$\frac{\partial J}{\partial y^{pred}} = -2(1 - ypred)$$

As $f_2$ is the linear activation, there is no transformation through $f_2$. Thus, $f_2(h_1 w_5 + h_2 w_6 + b_3) = h_1 w_5 + h_2 w_6 + b_3$. Thus:

$$y^{pred} = o_1 = f_2(h_1 * w_5 + h_2 * w_6 + b_3) = h_1 w_5 + h_2 w_6 + b_3$$

Thus:

$$\frac{\partial \boldsymbol{y^{pred}}}{\partial \boldsymbol{h_1}} = \frac{\partial f_2(h_1 w_5 + h_2 w_6 + b_3)}{\partial h_1} = \frac{\partial (h_1 w_5 + h_2 w_6 + b_3)}{\partial h_1} = w_5$$

Similarly, since:

$h_1 = f_1(x_1 * w_1 + x_2 * w_2 + b_1) = x_1 w_1 + x_2 w_2 + b_1$
(as $f_1$ is a linear activation function)

Thus:

$$\frac{\partial \boldsymbol{h_1}}{\partial \boldsymbol{w_1}} = \frac{\partial (x_1 w_1 + x_2 w_2 + b_1)}{\partial w_1} = x_1 = 2$$

Let initialize all the weights w to 1 and all the biases b to 0. Then:

$$\frac{\partial \boldsymbol{y^{pred}}}{\partial \boldsymbol{h_1}} = w_5 = 1$$

$$h_1 = f_1(x_1 w_1 + x_2 w_2 + b_1) = x_1 w_1 + x_2 w_2 + b_1 = 2 * 1 + 3 * 1 = 5$$

$$h_2 = f_1(x_1 w_3 + x_2 w_4 + b_2) = x_1 w_3 + x_2 w_4 + b_2 = 2 * 1 + 3 * 1 = 5$$

Thus:

$y^{pred} = o_1 = f_2(h_1 * w_5 + h_2 * w_6 + b_3) = h_1 w_5 + h_2 w_6 + b_3 = 5*1+5*1 = 10$

Thus:

$$\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{y^{pred}}} = -2\left(1 - y^{pred}\right) = -2(1 - 10) = 18$$

Thus:

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial y^{pred}} * \frac{\partial y^{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} = 18 * 1 * 2 = 36$$

Then the new $w_1' = w_1 - \epsilon * 36$. Let say the learning rate $\epsilon = 0.01$, then $w_1' = 1 - 0.01 * 36 = 0.64$.

After obtaining $w_1'$, we do the same process as above repeatedly (e.g. calculating $\frac{\partial J}{\partial w_1}$) until obtaining the optimal $w_1^*$; optimal $w_1^*$ is the value such that $\frac{\partial J}{\partial w_1^*}$ close to zero. The optimal $w_2^*$, $w_3^*$, $w_4^*$, $w_5^*$, $w_6^*$, $b_1^*$, $b_2^*$ and $b_3^*$ will be found with the same process for finding optimal $w_1^*$ as above. With the data containing only one example, this procedure also illustrates the process of stochastic gradient descent algorithm with the minibatch of one example.

### 3.1.4 Activation function

As discussed, the activation function can appear in both the hidden layer and the output layer. Such activation function can be a linear transformation function or nonlinear transformation function. The neural network is simply a linear model if all its layers only consist of linear transformation functions. The neural network illustrated in Section 3.1.3 turns into a linear regression model since its layers only consist of linear activation functions. On the other hand, when the problem requires solving the nonlinear relationship between the input and the output, the neural network layer must consist of a nonlinear transformation function. This turns the deep neural into a nonlinear model. There are different types of activation functions. The types of activation functions for the hidden layer and the output layer are chosen based on different principles.

As the choice of the activation function in the hidden layer affects the performance of the neural network model, the activation function for the hidden layer will be chosen if that activation function gives the best performance to the neural network model.

The activation function in the output layer is chosen based on the type of prediction problems. To recall, two main types of prediction problems in deep learning are regression and classification problems; classification problem consists of binary classification and multi-class classification.

3.1.4.1 Activation function in the hidden layer

Some common nonlinear activation functions for the hidden layer are rectified linear unit, logistic sigmoid and hyperbolic tangent. Although the principles of choosing an activation function in the hidden layer have been an active point of discussion in the literature, yet there have been no uniform theoretical guidelines. Thus, it is difficult to determine which type of activation function in the hidden layer gives the best performance for the models. However, in practice, the rectified linear unit is the activation function widely used and accepted in the hidden layer. The thesis will go through the basic concepts of the three said activation functions and explain why the rectified linear unit is preferred in the hidden layer.

a) Logistic sigmoid and hyperbolic tangent

Traditionally, logistic sigmoid function (or sigmoid function in short) and hyperbolic tangent function (or tanh function in short) are widely used in the hidden layers of the neural network.

Sigmoid function f(z) maps z into the value ranging from 0 and 1. The formula of the logistic sigmoid function is given as:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.32}$$

In tanh function, the value z is transformed to a value ranging from -1 to 1. The tanh function is mathematically defined as:

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.33}$$

The curves of both functions are visualized in Figure 9:

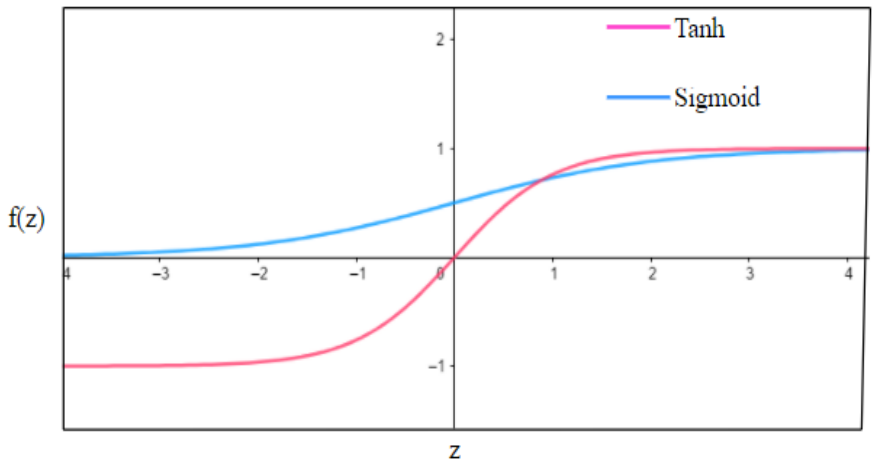


**Figure 9. Tanh and Sigmoid functions**

We have the derivatives of sigmoid function and tanh functions as:

$$f'(z) = \left(\frac{1}{1+e^{-z}}\right)' = f(z) * (1 - f(z)) \tag{3.34}$$

$$f'(z) = \left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)' = 1 - (f(z))^2 \tag{3.35}$$

From Figure 9, we can see that the sigmoid function saturates to 1 if the value of z is very high or saturates to 0 if the value of z is very low. Thus, from equation (3.34), when the value of z is very large and very small, the derivative of the sigmoid function is zero.

Similarly, the tanh function saturates to 1 or -1 when z is very large or very small, respectively. Thus, from equation (3.35), the derivative of the tanh function is also zero when z is very large or very small.

Thus, once saturated, both the sigmoid function and the tanh function challenge the gradient-based learning to determine which direction the weight parameters should move to update the weights. This limitation is called the **vanishing gradient problem**

preventing deep networks from learning effectively. This problem of the sigmoid function and the tanh function discourages users from using them in the hidden layer nowadays.

b) Rectified linear unit

Rectified linear unit (or ReLu in short) is mathematically defined as f(z) = max(0, z). Visually, ReLu function is described in Figure 10.

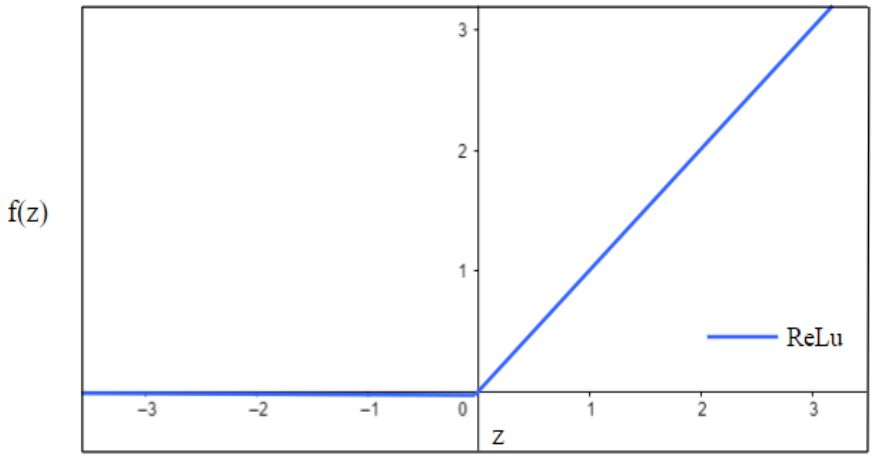


**Figure 10. ReLu function**

From the formula of ReLu and Figure 10, we can see that ReLu equals z for all positive values of z and equals 0 for all negative values of z. This implies several advantages of using ReLu in the neural network. First, using ReLu function in the hidden layer is easy for optimization because it is so similar to a linear function and requires no complicated math. Second, ReLu does not get saturated when z gets large or small. Thus, ReLu does not have the vanishing gradient problem like sigmoid or tanh functions.

However, being zero for all negative z values can cause the potential problem that gradient-based methods cannot learn when z is negative. This is called the “dying ReLU” problem. In practice, gradient-based learning still regularly performs well for the neural network models using ReLu. However, if the “dying ReLU” problem does happen, one option to solve the problem is to lower the learning rate.

Another choice for solving the "dying ReLU" problem is using variants of ReLu. Leaky ReLu is one common variant of ReLu. In Leaky ReLu, when z is negative, f(z) = $\alpha * z$ (instead of 0, like in original ReLu), where $\alpha$ is a small number, let say 0.01, for example. Leaky ReLu removes the zero-slope parts of the original ReLu solving the "dying ReLU" problem. Leaky ReLu can be visualized as:

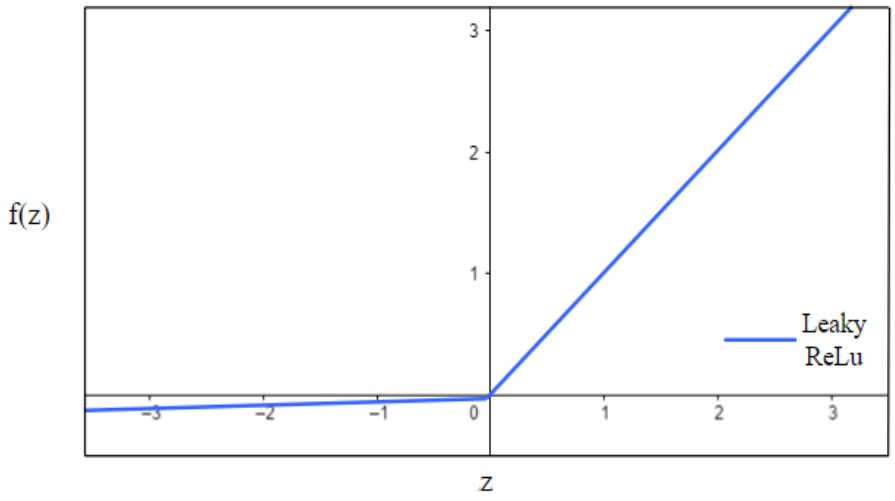


**Figure 11. Leaky ReLu function**

3.1.4.2 Activation function in the output layer

As discussed above, the activation function in the output layer is chosen based on the type of prediction problems. From Section 3.1.1, we know that the choice of the cost function is directly related to the prediction problems. Thus, the type of the prediction problem, the choice of activation function in the output layer and the cost function are closely connected. The most common activation functions in the output layer are linear, sigmoid and softmax.

a) Sigmoid and softmax output activation function

As introduced earlier in Section 3.1.4.1, the sigmoid function produces values ranging from 0 to 1 i.e. the sigmoid function converts the input z into one that can be interpreted as a probability. Thus, the sigmoid function is commonly used in the output layer when the problem is the binary classification problem as the binary classes are represented as either 0 or 1. Specifically, the class is predicted as 0 if the output layer returns a

value smaller than 0.5; otherwise, the model gives the prediction of 1. The softmax activation function produces discrete values ranging from 0 and 1, and all the values sum up to 1. As such, this type of activation function is suitable for multi-class classification. The softmax function is given as $f(\mathrm{z_i}) = \frac{e^{z_i}}{\sum_j^K e^{z_j}}$, where $z_i$ is the input vector's element and K is the number of classes in the multi-class classifier.

b) Linear output activation function

The linear activation function in the output layer does not transform the values derived from the hidden layer i.e. the values from the hidden layer are exactly themselves in the output layer. Since the output variable of the regression problem is a numerical value, the linear activation function is typically used in the output layer when the prediction problem is a regression problem. The cost function in this type of problem is either cross-entropy or MSE as discussed in Section 3.1.1.

## 3.2 Autoencoder

**Autoencoders** are an unsupervised learning method using the neural network model. The autoencoder is known as the unsupervised learning algorithm as the output used for training the neural network model is also its input. Thus, the autoencoders can be viewed as a special case of feedforward networks that can be trained with all the same processes and techniques such as minibatch gradient descent, back-propagation, etc., introduced in the earlier sections. Autoencoders are often used as a dimensionality reduction or feature-extracting tool. In autoencoder terminology, the neural network can be viewed as an encoder-decoder architecture. The encoder part compresses the high dimensional input to a new representation with a lower dimension (**latent state representation**). The layer that does such dimensionality-reduced task is called the **bottleneck layer**. Then latent state representation in the bottleneck layer is reconstructed to the original-high-dimensionality data in the decoder part.

Figure 12 describes the shallow architecture autoencoder with a single hidden layer, in which the autoencoder task is done by one single layer encoder and one single layer decoder. Given a training set containing m examples $\{x^{(1)},x^{(2)},\ldots,x^{(m)}\}$ and each

example has n features, thus an example of training set $X = [x_1, x_2, \ldots, x_n]$. After training through the shallow autoencoder, the output is given as $X' = [x'_1, x'_2, \ldots, x'_n]$. Let $Z_l$ denote the information generated from the $l^{th}$ layer of the neural network i.e. $Z_l$ is extracted feature from the $l^{th}$ layer, so $X = Z_0$. Then the shallow autoencoder model is explicitly given as:

$$Z_1 = f_1(W_1 X + b_1), \tag{3.36}$$

$$\widehat{X'} = f_2(W_2 Z_1 + b_2). \tag{3.37}$$

where W is weight matrices, and b is the bias vector.

As the problem in autoencoder is the regression problem, the cost function in autoencoder is $L_2$ loss - the concept that we discussed in Section 3.1.1. It is the two-norm difference between the input vector and the output vector over m examples (reconstruction error). Let $L_{2j}$ is the reconstruction error between the original feature $x_j$ and the reconstructed $x'_j$ ($x_j$ and $x'_j$ are the vectors with the length of m). Then the reconstruction error of the feature $x_j$ is given as:

$$L_{2j} = \|x_j - x'_j\| = \sqrt{(x_j^1 - x'^1_j)^2 + (x_j^2 - x'^2_j)^2 + \ldots + (x_j^m - x'^m_j)^2} = \sqrt{\sum_{i=1}^{m} (x_j - x'_j)^2} \tag{3.38}$$

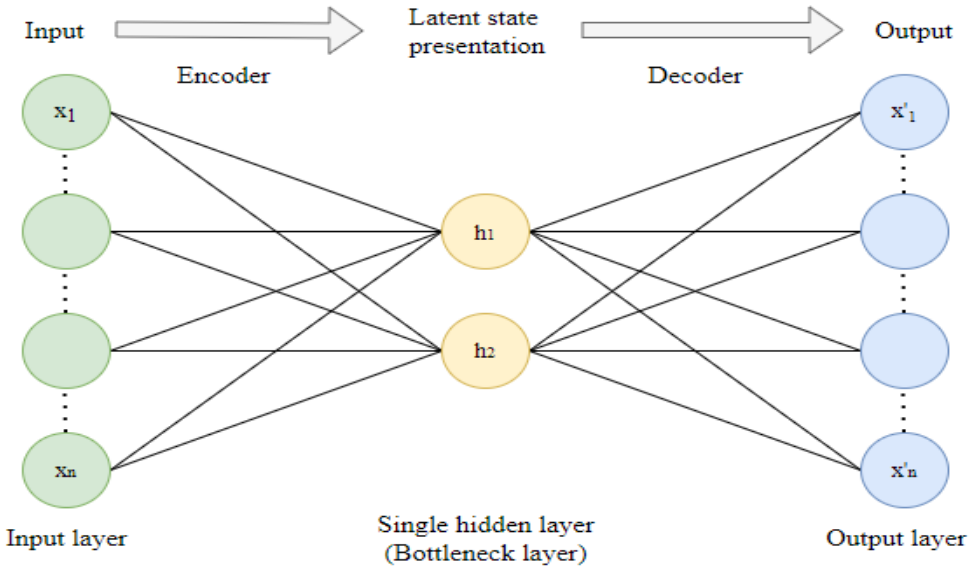


**Figure 12. Shallow autoencoder**

In fact, using deep encoders and decoders brings many advantages. The depth of the neural network (or autoencoder) can exponentially reduce the amount of training data required to learn some functions (Goodfellow, Bengio & Courville 2016, pp. 508-509). Experimentally, autoencoders with deep architecture perform compression much better than those with shallow architecture (Hinton and Salakhutdinov, 2006). The example of deep autoencoders is given in Figure 13, which has a two-layer encoder and a two-layer decoder. The decoded X' in the deep autoencoder is given as:

$$\widehat{X}' = f_L(W_L f_{L-1}(\dots W_2 f_1(W_1 X + b_1) + b_2 \dots) + b_L). \quad (3.39)$$

where L is a number of layers of the deep autoencoder.

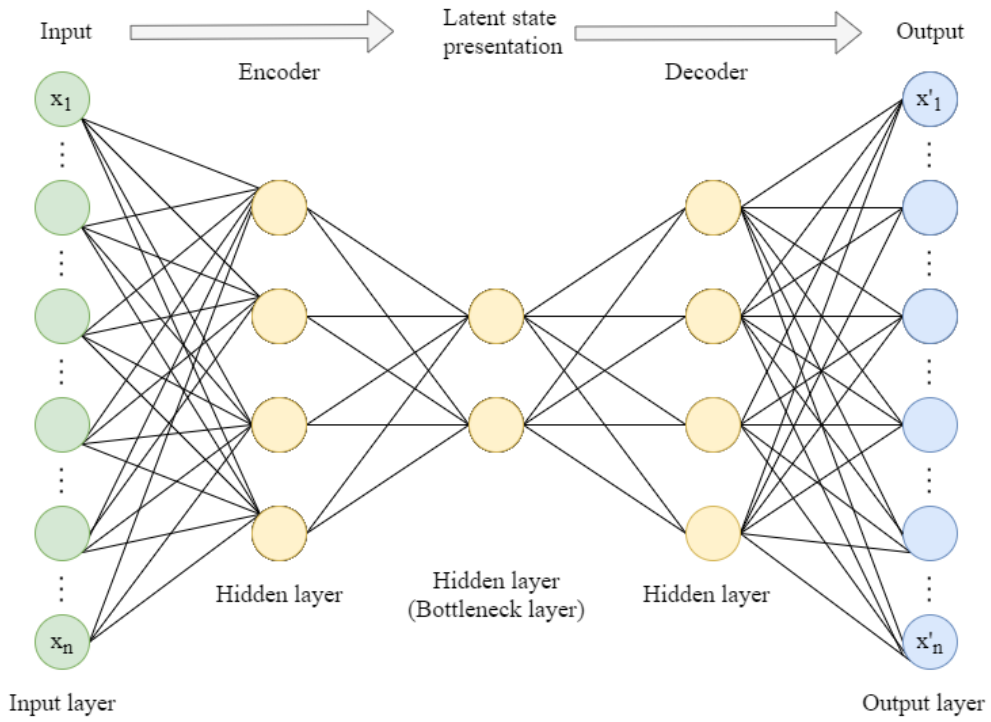


**Figure 13. Deep autoencoder**

## 3.3 Relation of autoencoder and CAPM

In the traditional Capital Asset Pricing Model (CAPM) by Sharpe (1963), the asset returns ($r_i$) are regressed on the benchmark returns ($r_b$):

$$r_i = \alpha + \beta r_b \tag{3.40}$$

However, as (3.40) is a linear model, it may miss the non-linearities between the benchmark returns and asset returns.

Considering applying autoencoder to investigate the relationship between the market index and stock constituent for solving the asset selection problem. The training data set contains m examples or m periods $\{x^{(1)},x^{(2)},\ldots,x^{(m)}\}$. Each example is the returns of n stock constituents on each trading period denoted as $x = [x_1, \ldots, x_n]$ (where n represents the number of the index's constituents or stock universe).

From the theoretical framework of the deep autoencoder, we can see that the information of the stock universe is compressed in the latent state presentation in the bottleneck layer. Thus, the latent state presentation can represent the market index information. After obtaining the market index information, the decoder part of the autoencoder network reflects the relationship between the market index and individual stock constituents. The decoder part does the same task as the CAPM model. However, while the CAPM model investigates the linear interaction between the returns of the benchmark and the asset, the deep autoencoder studies the nonlinear relationship between them.

We can apply the nonlinear relationship investigation of deep autoencoder to solve the asset selection problem in index tracking. The reconstruction error $\sqrt{\sum_{i=1}^{m}(x_j - x'_j)^2}$ can measure the similarity of a stock constituent with the market index, in which the stock with smaller reconstruction errors will have more common information with the market index. This investigation constructs a portfolio that can mimic the index structure.

### 3.4 Regularization in machine learning

The desired outcome in model training is a good performance in the test data. However, the model with good in-sample performance can experience poor out-of-sample performance due to the overfitting problem. The overfitting happens because the models attempt too hard to capture the noise in the training data which does not represent the true properties of the data. Regularization is a technique making the training models more flexible, thereby preventing the overfitting problem.

There are different regularization methods. One common method is $L_2$ regularization (or ridge regression) introduced by Hoerl and Kennard (1988). The $L_2$ regularization method introduced a more stable cost function than least squared errors by including a shrinkage quantity given as:

$$w^* = argmin_w \|y(w)^{pred} - y^{true}\|_2^2 + \lambda\|w\|_2^2 \tag{3.41}$$

The training model now attempts to minimize the cost function added a penalty or a regularizer $\lambda\|w\|_2^2$ , where λ is the tuning hyperparameter that is priorly chosen. The λ controls the flexibility level of the training model. When λ = 0, the penalty term has no effect, then the cost function turns into least squared errors which can cause the overfitting problem. The greater λ values give smaller estimated weights. When λ goes to ∞, the impact of the penalty term gets larger, and the estimated weights of some specific features approach zero, making the model more parsimonious.

The new problems come up: how to choose the good hyperparameter λ?. The cross-validation approach is used to solve this problem. One common method in the cross-validation approach is the validation set method. In the **validation set method**, we need a **validation set** separated from **the training set** to examine how well the λ value works. Figure 14 illustrates how the dataset is divided in the validation set method. The dataset is divided into three groups: training set looking for λ (blue set), validation set looking for λ (orange set) and test set looking for the out-of-sample performance of the model (green set). First, we try different λ values in the blue set to generate different training models (each model is associated with each λ). Then we examine how well each model performs in the orange set. The λ will be chosen if its model obtains the smallest error in the validation set. Once the λ has been chosen for the model, we use the yellow set (blue set + orange set) for model training. Then this training model will be tested in the green set to obtain the out-of-sample performance.

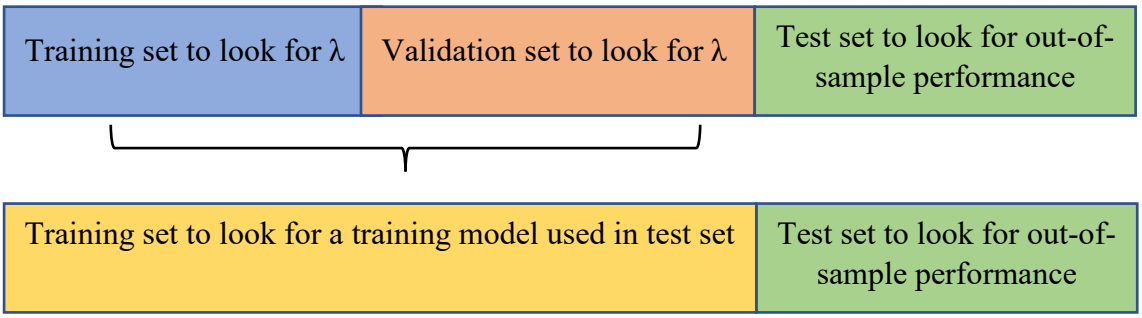


**Figure 14. Validation set method**

# 4 DATA AND METHODOLOGY

## 4.1 Data

The research first lists stock constituents contained in the S&P500 index and then obtains their weekly closing prices, which are used as the input for the autoencoder phase. The weekly closing prices of stock constituents and S&P500 index in the 9-year period from 1 January 2012 to 31 December 2020 are obtained from the Yahoo! Finance website's data. To avoid missing data and ensure the empirical results are reliable, the principles of obtaining the stock constituents for the autoencoder phase are:

(i) The data of obtained stock constituents must be present during the whole investigated period to ensure there is no missing data; and
(ii) Obtained stock constituents keep unchanged their names during the whole investigated period to ensure no mergers, no acquisitions, etc. have been made; and
(iii) Obtained stock must be constituted in the S&P500 index during the whole investigated period to ensure that the obtained stocks have not been ejected from the constituents of the S&P500 index.

With the said principles, the final number of obtained stock constituents is 463 (stock universe).

## 4.2 Methodology

As discussed in Section 2, the current methods under the joint approach are not optimal for tracking index as they often require heavy computation or are even irrelevant due to the weight constraint in index tracking. Thus, the two-step approach offers more applicable methods with less computational costs. The methodology is designed to construct a portfolio that can (i) follow the trends of the market index; and (ii) produce an excess return over the market index (index beating).

As introduced earlier, autoencoders can extract features of the input to the latent state presentation which can be used to discover the dataset's structure. Thus, autoencoder

is a prominent method for select a subset of stocks mimicking the index market structure. Furthermore, the deep autoencoder can provide better performance than the shallow one. Therefore, the research designs the deep architecture for the autoencoder model to select a subset of stock tracking the market index.

The study did use the returns of stock constituents to train in the deep autoencoder model; however, the model did not give the desired results. The very slight differences between return values may make the gradient descent algorithm failed in finding the reliable local optimum of the cost function. Thus, alternatively, the study trains closing prices of 463 stocks with the deep autoencoder model to select stock from the stock universe for the tracking portfolio. The training data set contains m examples $\{x^{(1)},x^{(2)},\ldots,x^{(m)}\}$ (where m = weekly periods in the training set). Each example is the closing prices of n stock constituents on each trading week denoted as $x = [x_1, \ldots, x_n]$ (where n = 463 representing the number of the stock universe).

Gradient descent algorithm can get stuck in finding the local minimum when values of one or more features are much larger than the rest. As normalization scales the features in a specific range, it can speed up the gradient descent algorithm i.e., making the convergence of the cost function easier than one without normalization (Troiano, Bhandari & Villa 2020, p. 151). Min-max normalization is one common normalization method used in data analysis, which rescales the original data into the range [a, b].

According to Troiano, Bhandari & Villa (2020, p. 151), closing prices of universe stocks are rescaled by min-max normalization in the [0, 1] range. For a given range [0, 1], the initial closing prices are rescaled as:

$$x_{norm(i,m)} = a + \frac{x_{i,m} - min(x_i)}{max(x_i) - min(x_i)}(b - a)$$
$$= 0 + \frac{x_{i,m} - min(x_i)}{max(x_i) - min(x_i)}(1 - 0) = \frac{x_{i,t} - min(x_i)}{max(x_i) - min(x_i)} \qquad (4.1)$$

where $x_{i,m}$ is the weekly closing price of stock *i* in week *m,* and $x_{norm(i,m)}$ is a normalized price of stock *i* in week *m.*

The architecture of the deep autoencoder model is designed as in Figure 15. The numbers of hidden layers and neurons in the architecture are decided after assessing the error of multiple model trials. The deep autoencoder has three hidden layers. The 1st hidden layer has eight neurons, the 2nd hidden layer (the bottleneck layer) has four neurons, and the last hidden layer has eight neurons. As ReLu has advantages over other activation functions discussed in Section 3.1.4, ReLu is chosen as the activation function in all three hidden layers. As the problem is the regression problem, the selected activation function for the output layer is the linear function (in line with Section 3.1.4). The input X is mapped with the output X' through the deep autoencoder with five layers, where the input layer is the 0th layer and the output layer is the 4th layer as follows:

$$\widehat{X'} = f_4(W_4 f_3(W_3 f_2(W_2 f_1(W_1 X + b_1) + b_2) + b_3) + b_4).$$

where W is weight matrices; b is the bias vector and f is the activation function.

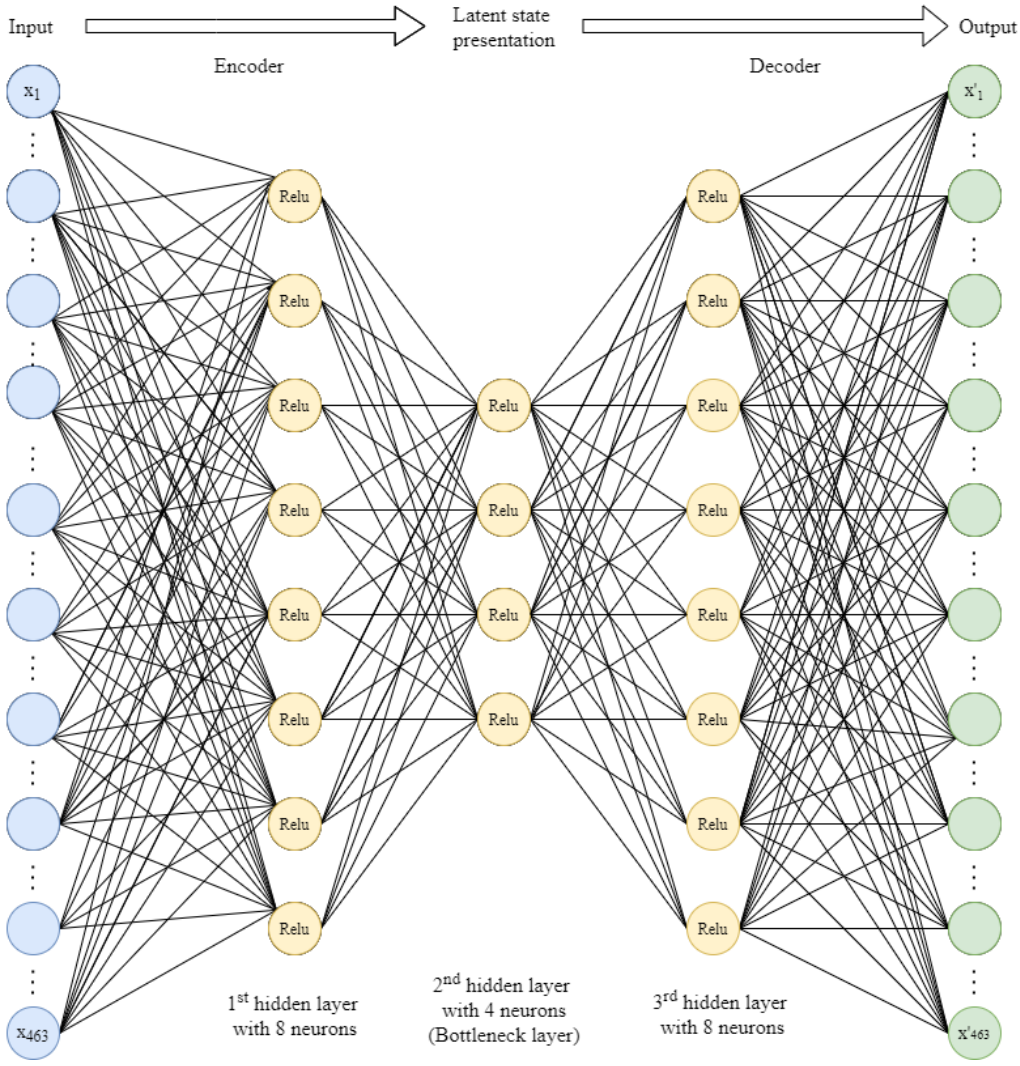


**Figure 15. Designed deep autoencoder model for stock selection**

As this is a regression problem, the cost function of the model is $L_2$ loss (or reconstruction error) (in line with Section 3.1.1). The $L_2$ loss or reconstruction error of the *j*-th stock is the total two-norm difference between the original normalized closing prices the reconstructed normalized closing prices over m periods:

$$L_{2j} = \|x_j - x'_j\| = \sqrt{\left(x_j^1 - x'^1_j\right)^2 + \left(x_j^2 - x'^2_j\right)^2 + \ldots + \left(x_j^m - x'^m_j\right)^2}$$

$$= \sqrt{\sum_{i=1}^{m}(x_j^i - x'^i_j)^2} \qquad (4.2)$$

The model is tuned with a learning rate = 0.01 and minibatch = 6.

After reconstructing X to get the output X', the reconstruction errors of 463 stocks are obtained. The stock with a small reconstruction error represents the proximity of its decoded version to its origin. The small reconstruction error of one stock also indicates the high similarity of such stock to the stock universe. In contrast, stocks with large construction errors share less common information with the stock universe. The research ranks the 463 stocks from one with the smallest reconstruction error (the most communal stock) to one with the largest reconstruction error (the least communal stock). According to Heaton et al., containing stocks with the same information does not provide more market information; thus, medium-communal stocks are not included in the portfolio. Therefore, the study constructs the tracking portfolios with the most communal stocks and the least communal stocks.

### 4.2.1 Index tracking with joint portfolio

The thesis first constructs the tracking portfolios with the capability of following the market trends. Three different tracking portfolios are constructed by combining the most communal stock and the least communal stocks (**tracking joint portfolio**). After assessing the portfolio performances in different sizes and combinations, the study decides to construct three portfolios containing 25 stocks (S25), 35 stocks (S35) and 45 stocks (S45), respectively. All S25, S35 and S45 contain the ten most communal stocks. The remaining components of S25, S35 and S45 are the 15, 25 and 35 least communal stocks, respectively.

Heaton et al. (2017b) did not discuss how assets were weighted in their research. After obtaining a subset of stocks for the tracking portfolio, they used selected stocks' returns as input and index returns as output to train the feedforward neural network model. As the input (stock returns) went through a nonlinear activation function in the hidden layers to map with the output (index returns), the training neural network model could not obtain the direct effect of stocks on the index. Furthermore, it is needed to put some constraints on the invested weights to make the model plausible in real life i.e. all the weights sum up to one and each weight must not be negative. However, no constraints were made in the feedforward neural network of Heaton et al., making their study less applicable in a practical sense.

Thus, differing from Heaton et al.'s study, this research implements the **validation phase** and the **calibration phase** for **stock weighting.** Invested weights of stock components are determined by solving a quadratic programming problem, in which the objective function for minimizing is tracking error variance. To avoid overfitting problem i.e. enhancing the out-of-sample performance, the research incorporates $L_2$ regularization term introduced in Section 3.4 into the objective function. The research also adds a constraint restricting short selling (non-negative stock weights) and a constraint that all weights sum up to unity. The quadratic programming problem is defined as follows:

$$w^* = argmin_w \|R_I - R_x w\|_2^2 + \lambda \|w\|_2^2 \quad (4.3)$$
$$\text{s.t. } \sum_{i=1}^{n} w_i = 1,$$
$$w_i \geq 0$$

where $R_I \in \mathbb{R}^m$ is a vector of index returns in m periods; $R_x = [R_1, \ldots, R_n] \in \mathbb{R}^{mxn}$ is the return matrix of n component stocks in m periods; $w = [w_1, \ldots, w_n] \in \mathbb{R}^n$ is a vector of stock weights (so that $R_x w$ is the portfolio return); and $\lambda \|w\|_2^2$ is a regularization term (regularizer).

The chronological order for stock weighting is conducted as follows: First, the arithmetic returns of the selected stocks (from autoencoder phase) and the index are calculated from the closing prices as:

$$r_{i,m} = \frac{x_{i,m} - x_{i,m-1}}{x_{i,m-1}} \quad (4.4)$$

where $r_{i,m}$ is the return of stock *i* or the index in week *m;* and $x_{i,m}$ is the weekly closing price of stock *i* or index in week *m*.

Next, we need to define the value of λ before solving the problem (4.3). The value of λ is determined in the **validation phase**. The 4-year set in the validation phase is divided into two subsets: a 3-year training set and a 1-year validation set. In the 3-year training set, the research trains the model with 30 different values of λ ranging from 0.001 to 0.03 (spacing between values is 0.001). As a result, the research has 30

training models corresponding to 30 values of λ for each portfolio. The research uses the 1-year validation set to evaluate how each model performs; the value of λ will be chosen if its model has the smallest error in the validation set.

After determining the appropriate value of λ, the research uses a 4-year set to solve the problem (4.3) (**calibration phase**). Finally, in the **testing phase**, the research tests the performance of the training model drawn from the calibration phase in the 1-year testing set.

### 4.2.2 Index beating with joint portfolio

The thesis next constructs the portfolios generating excess returns over the index. Three beating portfolios are constructed by containing the same stocks as the three joint tracking portfolios. However, to beat the market, the invested weights of the three beating portfolios are determined by using index returns $R_I$ added 2% in model training instead of original $R_I$ (**joint beating portfolio**). The purpose of this added 2% is to look for the invested weights generating portfolio returns higher than the index returns. The problem (4.3) turns to:

$$w^* = argmin_w \|(R_I + 2\%) - R_x w\|_2^2 + \lambda \|w\|_2^2 \quad (4.5)$$
$$\text{s.t. } \sum_{i=1}^{n} w_i = 1,$$
$$w_i \geq 0.$$

### 4.2.3 Index beating with sparse portfolio

While constructing three joint portfolios beating the market, it is observed that the proportion of the most communal stocks' invested weights are dominant over the least communal stocks'. This implies that it is possible to construct a portfolio containing only the most communal stocks beating the market. Thus, the research next constructs a portfolio beating the market containing only the ten most communal stocks (the sparse portfolio). The research constructs the sparse portfolio with the same process as the joint beating portfolio set in Section 4.2.2.

The procedure of the four phases in the 5-year dataset is briefly described in Figure 16. This process continues for five years to obtain five different yearly performances to affirm the out-of-sample performance of the portfolios. Continuous dataset arrangement for training and testing during the whole 9-year period is described in Figure 17.

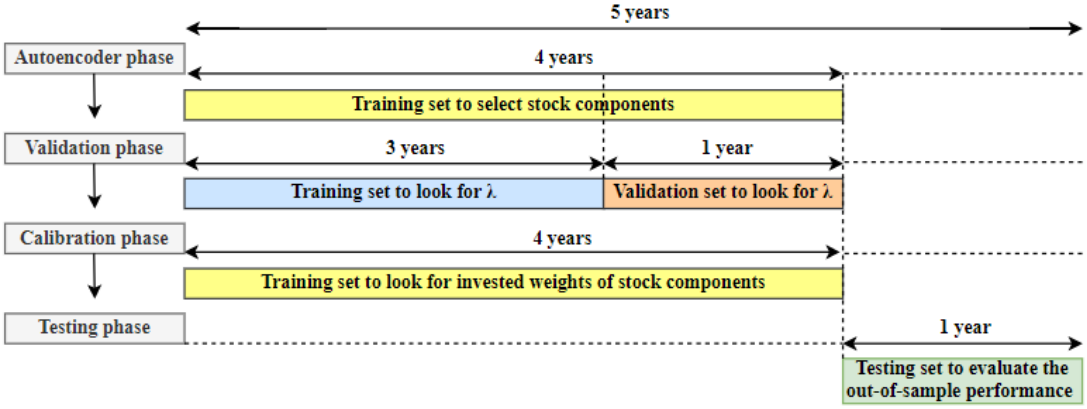


**Figure 16. The procedure of autoencoder, validation, calibration and testing phases in a 5-year period**

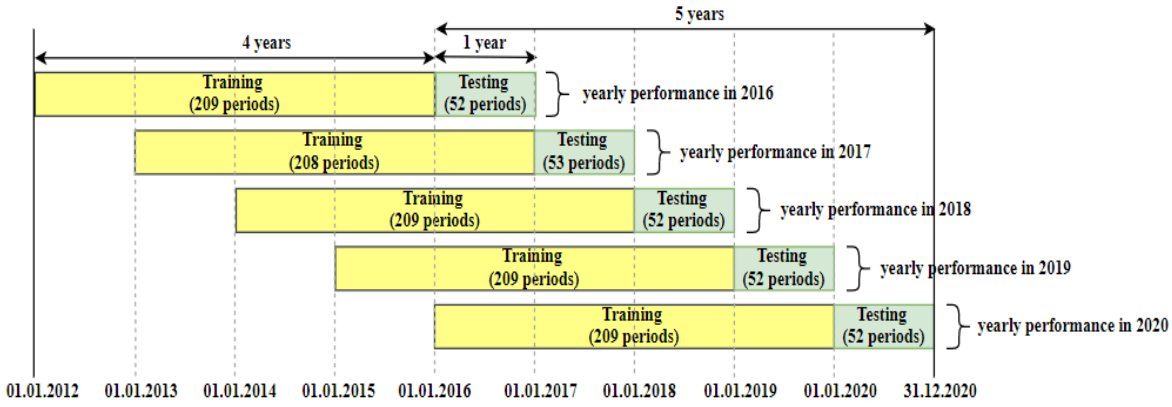


**Figure 17. Continuous dataset arrangement for training and testing during the entire 9-year dataset**

### 4.2.4 Performance measurement

We evaluate the performances of constructed portfolios with two measurements: cumulative abnormal return and beta. While cumulative abnormal return measures the

portfolio performance in generating excess returns over the market index, beta measures the portfolios' volatility relative to the market index i.e. risk measuring.

The equation for calculating the cumulative abnormal return is:

$$CAR = \sum_{j=1}^{m}(R_x w - R_I) \tag{4.6}$$

where $R_I \in \mathbb{R}^m$ is a vector of index returns in m periods of the test set; $R_x = [R_1, \ldots, R_n] \in \mathbb{R}^{mxn}$ is the return matrix of n component stocks in m periods of the test set; w = $[w_1, \ldots, w_n] \in \mathbb{R}^n$ is a vector of stock weights (so that $R_x w$ is the portfolio return);

Beta is calculated as:

$$Beta = \frac{Covariance(R_P, R_I)}{Variance(R_I)} \tag{4.7}$$

where $R_I$ and $R_P$ are the index return and the portfolio return in the test set, respectively.

# 5 EMPIRICAL RESULTS

As described in the methodology section, the research implements the process continuously for five years to obtain five different yearly performances. As examining the out-of-sample performance of the portfolio in the large drawdown of the market in 2020 caused by Covid-19 is interesting, the research specifically represents the four-phase process to obtain the portfolio performance in 2020. The portfolio performances in other prediction years are briefly represented after that.

In the autoencoder phase, the stocks were sorted from the lowest reconstruction errors to the largest ones to determine their communal rankings. Figures 18 and 19 describe how the decoded versions of the most communal stock and the least communal stock fit their original data in the 4-year set from 01.01.2016 to 31.12.2019, where MSFT and EIX share the most and the least common information with the stock universe, respectively. Figures 20 and 21 show the ten most communal stocks and the 35 least communal stocks, respectively, which are used to construct three different portfolios described in Section 4.2.

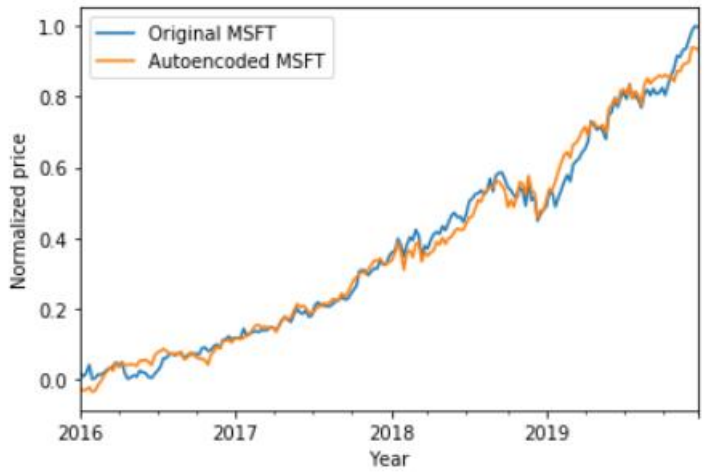


**Figure 18. Original and decoded versions of the most communal stock**

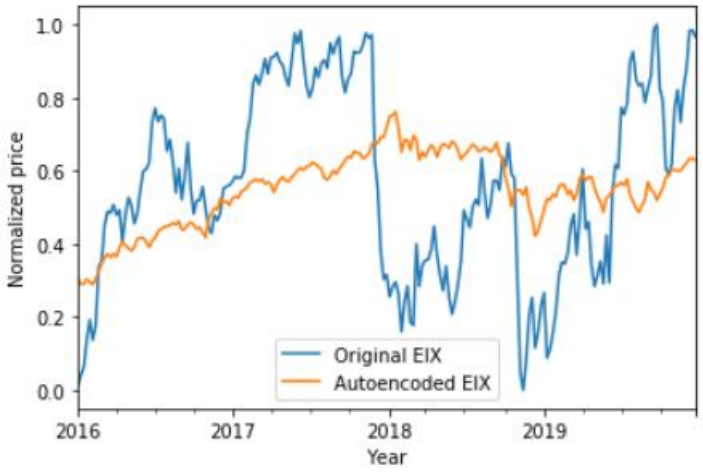


**Figure 19. Original and decoded versions of the least communal stock**

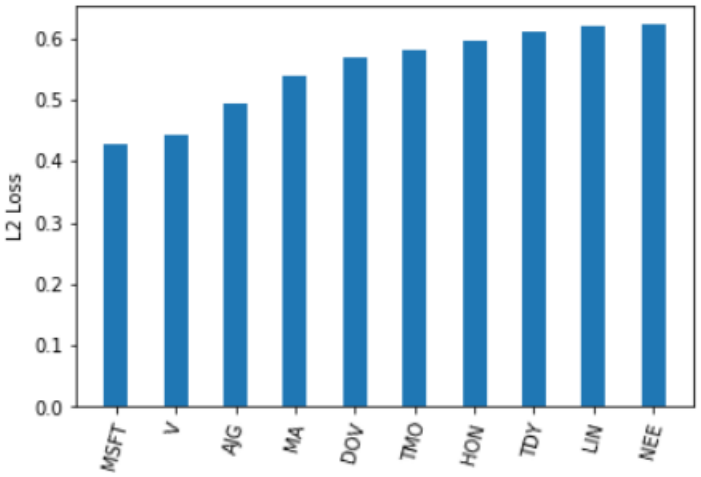


**Figure 20. The ten most communal stocks based on reconstruction errors**

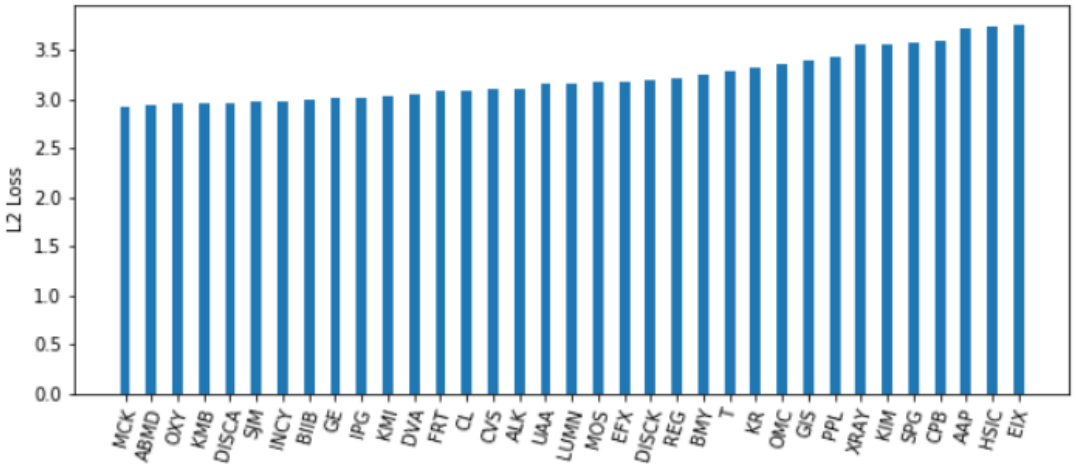


**Figure 21. The 35 least communal stocks based on reconstruction errors**

## 5.1 Index tracking with joint portfolio

### 5.1.1 Validation phase

After the autoencoder phase, the three portfolios are constructed from the selected stocks. Then we have three accordingly different models in the validation phase, namely as S25, S35 and S45. As lambda in $L_2$ regularization method controls the out-of-sample performance of the three models, this phase aims to look for the most appropriate lambda values over 30 values ranging from 0.001 to 0.03 (with the spacing between values is 0.001) for the following problems:

Portfolio S25: $w^* = argmin_w \|R_I - R_{x\_S25} w\|_2^2 + \lambda \|w\|_2^2$ (5.1)

Portfolio S35: $w^* = argmin_w \|R_I - R_{x\_S35} w\|_2^2 + \lambda \|w\|_2^2$ (5.2)

Portfolio S45: $w^* = argmin_w \|R_I - R_{x\_S45} w\|_2^2 + \lambda \|w\|_2^2$ (5.3)

s.t. $\sum_{i=1}^{n} w_i = 1$,

$w_i \geq 0$

In the 3-year training set of validation phase from 01.01.2016 to 31.12.2018, the thesis substitutes each value of lambda to the three models (5.1), (5.2) and (5.3) to look for the model parameters. Then, each model will have 30 sub-models corresponding to 30 lambda values. The value of lambda is picked if its sub-model has the smallest error in the validation set. Based on the performances of 30 sub-models in the 1-year validation set from 01.01.2019 to 31.12.2019 illustrated in Figure 22, the most appropriate lambda values for S25, S35 and S45 are 0.021, 0.019 and 0.018, respectively.

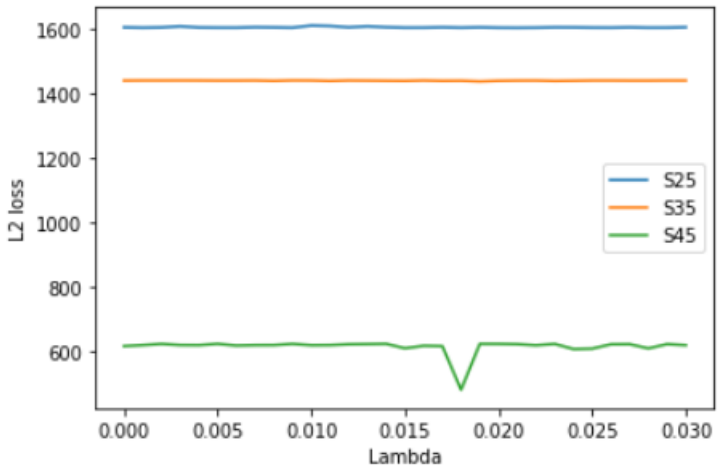


**Figure 22. Performances of 30 lambda values in the validation set**

### 5.1.2 Calibration phase

After selecting the appropriate lambda values for the three models S25, S35 and S45. The weights of stock components in the three portfolios are computed by solving the following minimization problems:

$$\text{Portfolio S25: } w^* = argmin_w \|R_I - R_{x\_S25}w\|_2^2 + 0.021\|w\|_2^2 \quad (5.4)$$

$$\text{Portfolio S35: } w^* = argmin_w \|R_I - R_{x\_S35}w\|_2^2 + 0.019\|w\|_2^2 \quad (5.5)$$

$$\text{Portfolio S45: } w^* = argmin_w \|R_I - R_{x\_S45}w\|_2^2 + 0.018\|w\|_2^2 \quad (5.6)$$

$$\text{s.t. } \sum_{i=1}^{n} w_i = 1,$$

$$w_i \geq 0$$

where $R_I$ and $R_x$ are index returns and stock components' returns over the 4-year training period from 01.01.2016 to 31.12.2019.

Figure 23 shows the invested weights of stock components of each portfolio. The cumulative returns of the index and the three portfolios in the training period are illustrated in Figure 24. We can see that the three portfolios can track the trend of the index return over the training period.

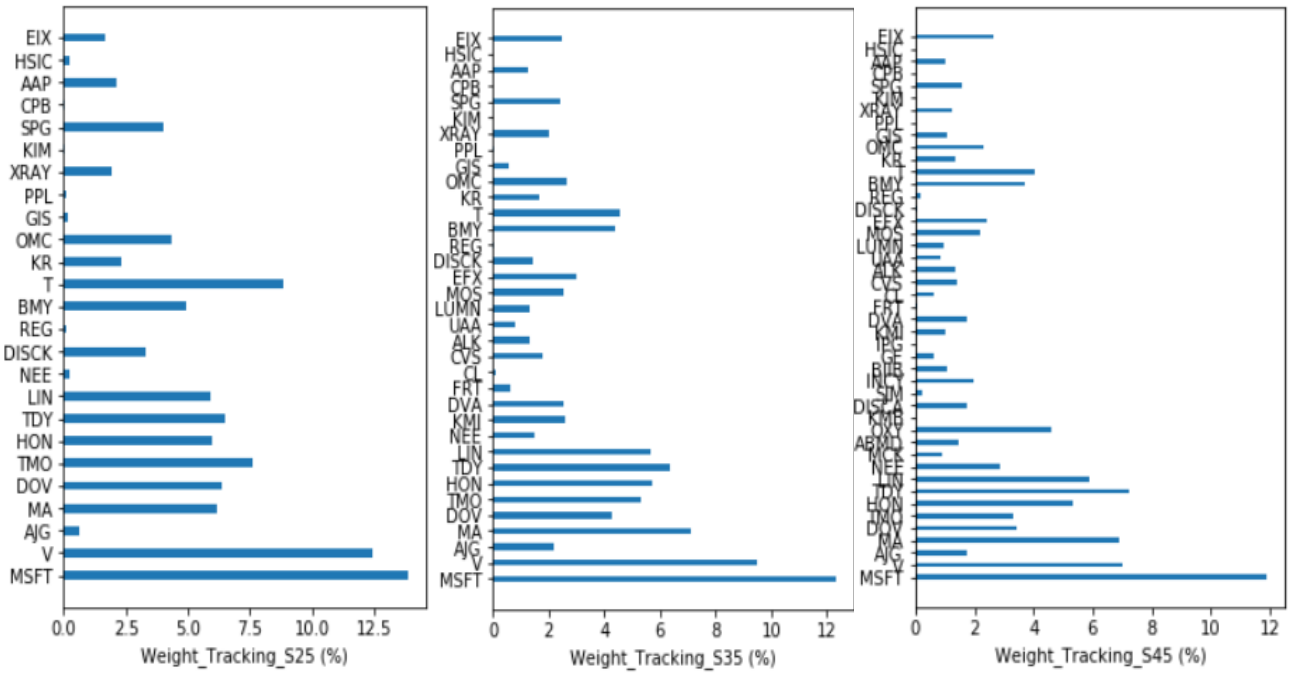


**Figure 23. Invested weights of stock components of the three tracking portfolios**


**Figure 24. Cumulative returns of the index and the three tracking portfolios in the training set**

### 5.1.3 Testing phase

We examine the out-of-sample performance of the three tracking portfolios i.e. their index-tracking ability in the test set. Figure 25 describes the out-of-sample performances of the three tracking portfolios in the 1-year period from 01.01.2020 to 31.12.2020. It is observed that all three tracking portfolios can follow the trends of the

market. We witness the large drawdown in March 2020 caused by Covid 19. The cumulative return lines of the three portfolios were relatively below the market index's after the large drawdown. Thus, tracking portfolios tend to underperform the market index in the period having a large drawdown.

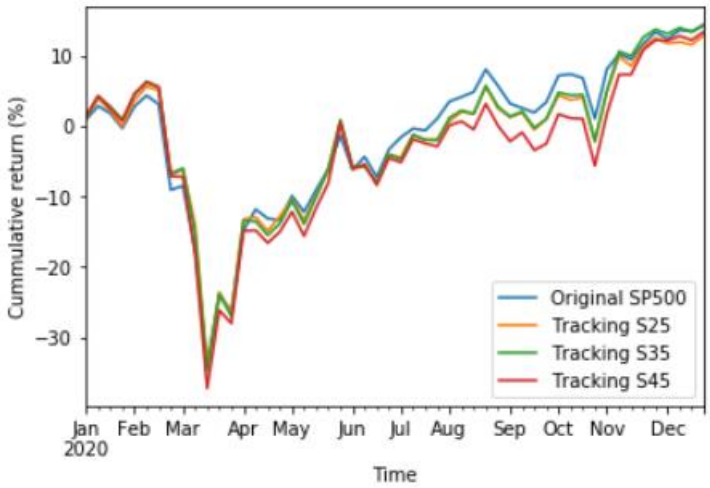


**Figure 25. Cumulative returns of the index and the three tracking portfolios in the test set**

## 5.2 Index beating with joint portfolio

The return lines of the three portfolios need to be lifted above the return line of the index to beat the market. Thus, the thesis does not use original index returns $R_I$ for model training, but $R_I$ added 2%. The problems (5.1), (5.2) and (5.2) turn to:

Portfolio S25: $w^* = argmin_w \|(R_I + 2\%) - R_{x\_S25}w\|_2^2 + \lambda\|w\|_2^2$ (5.7)

Portfolio S35: $w^* = argmin_w \|(R_I + 2\%) - R_{x\_S35}w\|_2^2 + \lambda\|w\|_2^2$ (5.8)

Portfolio S45: $w^* = argmin_w \|(R_I + 2\%) - R_{x\_S45}w\|_2^2 + \lambda\|w\|_2^2$ (5.9)

s.t. $\sum_{i=1}^{n} w_i = 1$,

$w_i \geq 0$

### 5.2.1 Validation phase

Like the validation phase in Section 5.1, the three most appropriate lambda values are selected based on the validation set method. However, this section looks for the lambda values for the three portfolio models by solving the problems (5.7), (5.8) and (5.9).

Figure 26 shows the performances of 30 sub-models corresponding to 30 lambda values in the validation set, where the most appropriate lambda values for S25, S35 and S45 are 0.025, 0.02 and 0.016, respectively.

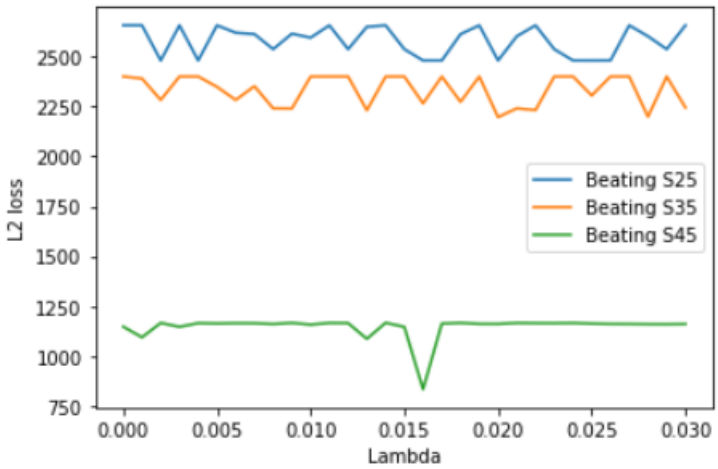


**Figure 26. Performances of 30 lambda values in the validation set**

### 5.2.2 Calibration phase

The study determines the invested weights of stock components by solving the following problem:

$$\text{Portfolio S25: } w^* = argmin_w \|(R_I + 2\%) - R_{x\_S25}w\|_2^2 + 0.025\|w\|_2^2 \quad (5.10)$$

$$\text{Portfolio S35: } w^* = argmin_w \|(R_I + 2\%) - R_{x\_S35}w\|_2^2 + 0.02\|w\|_2^2 \quad (5.11)$$

$$\text{Portfolio S45: } w^* = argmin_w \|(R_I + 2\%) - R_{x\_S45}w\|_2^2 + 0.016\|w\|_2^2 \quad (5.12)$$

$$\text{s.t. } \sum_{i=1}^{n} w_i = 1,$$

$$w_i \geq 0$$

Figure 27 shows the invested weights of stock components of the three portfolios for index beating. It is noticed that the ten most communal stocks account for the largest weight proportion while the least communal stocks' weights are insignificant. This observation leads to the analysis of beating the market with only ten most communal stocks illustrated in Section 5.3. Figure 28 shows that the return lines of the three portfolios can track the trend of the market and are well above the return line of the

index over the training period. We next examine their out-of-sample performances in the test set.

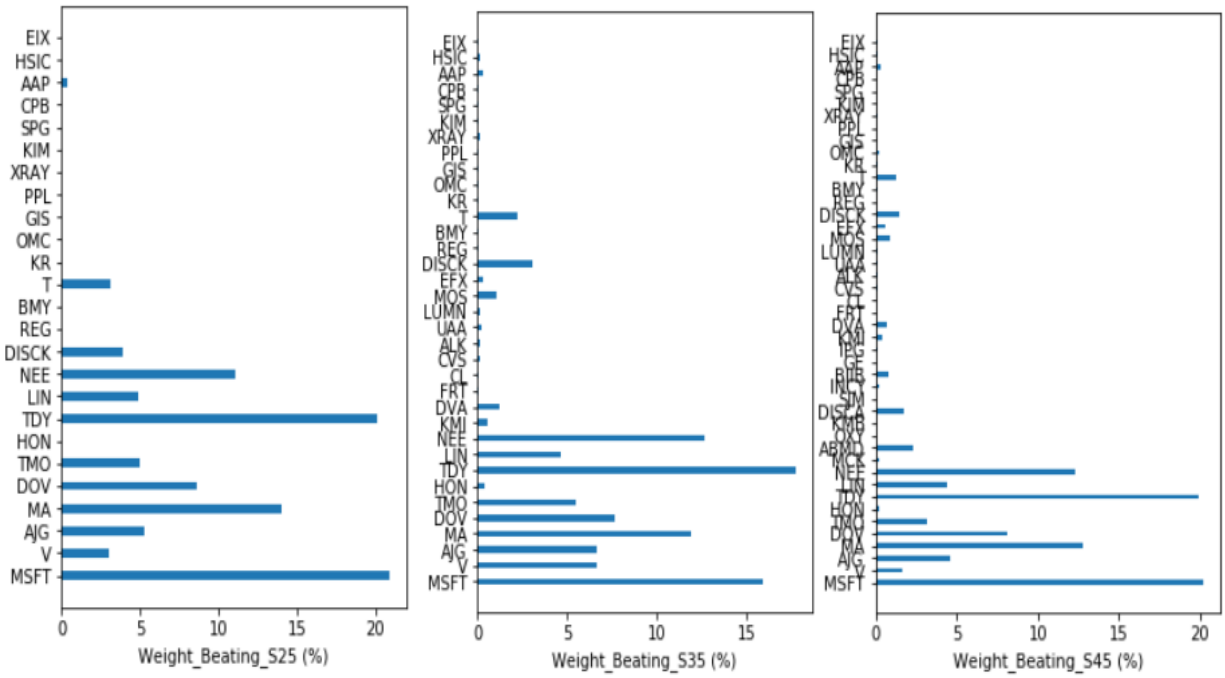


**Figure 27. Invested weights of stock components of the three beating portfolios**


**Figure 28. Cumulative returns of the index and the three beating portfolios in the training set**

### 5.2.3 Testing phase

This phase tests the out-of-sample performances of the three beating portfolios described in Figure 29. We can see that all the three return lines of the beating portfolios were well above the return line of the market index in the test period as

expected. This indicates that beating portfolios did constantly beat the market index over a 1-year-period despite the large drawdown of the market in March 2020 due to Covid-19. As of the end of 2020, the cumulative return of the index was 14.3% while the cumulative returns of the S25, S35, S45 were 21.1%, 21.1% and 22.5%, respectively. Replacing original index returns by index return added 2% in training set does construct portfolios generating returns higher than index returns in the test set. Another comment can be made that although the three portfolios have different sizes, the portfolio with a smaller number of stocks does not perform less well than the one with a larger number of stocks.

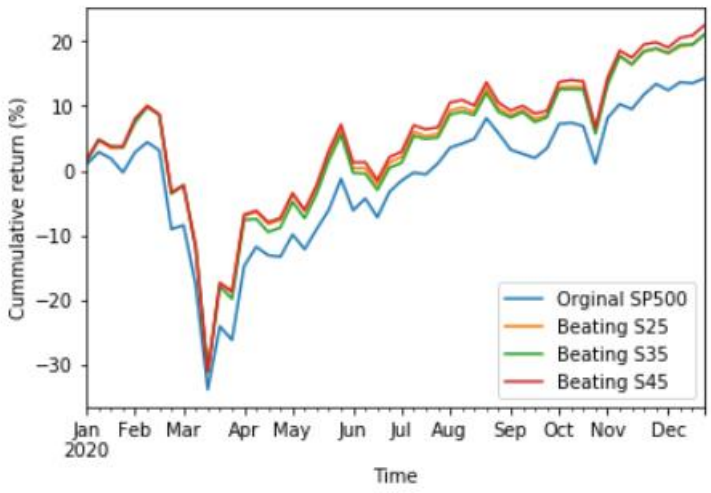


**Figure 29. Cumulative returns of the index and the three beating portfolios in the test set**

## 5.3 Index beating with sparse portfolio

As discussed earlier in Section 5.2, the ten most communal stocks account for most of the invested weight proportion described in Figure 27. This induces the construction of a beating portfolio containing only ten most communal stocks (sparse portfolio). The process of constructing the sparse portfolio S10 is implemented the same as joint beating portfolios' set in Section 5.2.

### 5.3.1 Validation phase

Basing on the performances of 30 lambdas in the validation set illustrated in Figure 30, the most appropriate lambda value for S10 is 0.024.

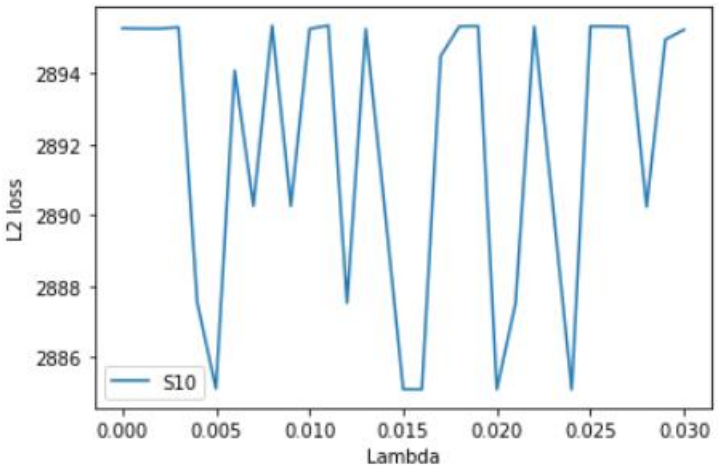


**Figure 30. Performances of 30 lambda values in the validation set**

### 5.3.2 Calibration phase

The invested weights of stock components of the sparse portfolio are provided by solving this:

Portfolio S10: $w^* = argmin_w \|(R_I + 2\%) - R_{x\_S10}w\|_2^2 + 0.024\|w\|_2^2$ (5.13)

s.t. $\sum_{i=1}^{n} w_i = 1$,

$w_i \geq 0$

Figure 31 shows the invested weights of stock components of the sparse portfolio while the in-sample performance of the sparse portfolio is described in Figure 32.

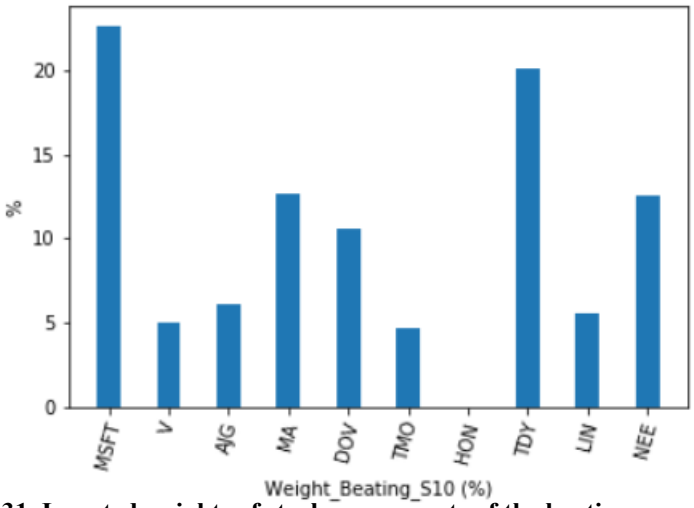


**Figure 31. Invested weights of stock components of the beating sparse portfolio**


**Figure 32. Cumulative returns of the index and the beating sparse portfolio in the training set**

### 5.3.3 Testing phase

Figure 33 shows that although the sparse portfolio contains only ten stocks, it could outperform the index constantly over the test set despite the large drawdown of the market in March 2020 due to Covid-19. As of the end of 2020, the cumulative return of the index was 14.3% while the cumulative return of the sparse portfolio was 23.8%.


**Figure 33. Cumulative returns of the index and the beating sparse portfolio in the test set**

The tables below report cumulative abnormal returns and beta values of the portfolios in five prediction years. The five annual performances are also visualized in the corresponding figures.

The results show that all the beating portfolios S10, S25, S35 and S45 consistently beat the market index in all five prediction years with the cumulative abnormal returns ranging from 3.8% to 12.8%. The beta values of all the beating portfolios in five prediction years were close to and below one (ranging from 0.83 to 1) except for the beating portfolios' beta values in 2020. In 2020, both tracking and beating portfolios were more volatile than the market index with beta values around 1.1. Thus, the constructed portfolios tend to be more volatile than usual in the period with large drawdowns.

The portfolio with a smaller number of stocks did not perform less well than the portfolio with a larger number of stocks. The sparse portfolio with ten stocks even had better performance than beating joint portfolios with 25, 35 and 45 stocks in four out of five years that are 2016, 2017, 2019 and 2020 in terms of cumulative abnormal returns. The beta value range of the sparse portfolio was similar with beating joint portfolios' from 0.83 to 1 (except for sparse portfolio's beta in 2020 with the value of 1.1). The outperformance of the sparse portfolio indicates that since the most communal stocks share the most common information with the index, they contain enough information to follow the market trends without the additional information from the least communal stocks.

Thus, the sparse portfolio with ten stocks can outperform the market index with acceptable riskiness. However, the sparse portfolio can be a risky investment in the large drawdown period.

**Table 2. Portfolio performance in 2016**

| | **SP500's cumulative return (%)** | **Portfolio's cumulative return (%)** | **Cumulative abnormal returns (%)** | **Beta** |
|---|---|---|---|---|
| Tracking S25 | | 15.67 | 6.56 | 0.97 |
| Tracking S35 | | 16.05 | 6.94 | 1.02 |
| Tracking S45 | | 18.1 | 8.99 | 0.96 |
| Beating S25 | 9.11 | 12.98 | 3.87 | 0.97 |
| Beating S35 | | 12.95 | 3.84 | 0.98 |
| Beating S45 | | 12.95 | 3.84 | 0.97 |
| Beating S10 | | 13.22 | 4.11 | 0.96 |

**Table 3. Portfolio performance in 2017**

| | SP500's cumulative return (%) | Portfolio's cumulative return (%) | Cumulative abnormal returns (%) | Beta |
|---|---|---|---|---|
| Tracking S25 | | 24.99 | 4.67 | 0.99 |
| Tracking S35 | | 24.39 | 4.07 | 0.97 |
| Tracking S45 | | 22.98 | 2.66 | 0.97 |
| Beating S25 | 20.32 | 29.73 | 9.41 | 0.85 |
| Beating S35 | | 28.47 | 8.15 | 0.98 |
| Beating S45 | | 28.65 | 8.33 | 0.89 |
| Beating S10 | | 27.74 | 7.42 | 0.83 |

**Table 4. Portfolio performance in 2018**

| | SP500's cumulative return (%) | Portfolio's cumulative return (%) | Cumulative abnormal returns (%) | Beta |
|---|---|---|---|---|
| Tracking S25 | | -6.80 | 1.21 | 0.96 |
| Tracking S35 | | -13.09 | -5.08 | 0.91 |
| Tracking S45 | | -9.41 | -1.40 | 0.94 |
| Beating S25 | -8.01 | -3.27 | 4.74 | 0.87 |
| Beating S35 | | -2.28 | 5.73 | 0.87 |
| Beating S45 | | -2.07 | 5.94 | 0.88 |
| Beating S10 | | -2.99 | 5.02 | 0.88 |

**Table 5. Portfolio performance in 2019**

| | SP500's cumulative return (%) | Portfolio's cumulative return (%) | Cumulative abnormal returns (%) | Beta |
|---|---|---|---|---|
| Tracking S25 | | 32.19 | 7.69 | 0.97 |
| Tracking S35 | | 33.48 | 8.98 | 0.92 |
| Tracking S45 | | 31.32 | 6.82 | 0.90 |
| Beating S25 | 24.50 | 36.58 | 12.08 | 0.96 |
| Beating S35 | | 37.00 | 12.50 | 0.94 |
| Beating S45 | | 36.50 | 12.00 | 0.92 |
| Beating S10 | | 37.28 | 12.78 | 1.00 |

**Table 6. Portfolio performance in 2020**

| | SP500's cumulative return (%) | Portfolio's cumulative return (%) | Cumulative abnormal returns (%) | Beta |
|---|---|---|---|---|
| Tracking S25 | | 12.97 | -1.33 | 1.09 |
| Tracking S35 | | 14.53 | 0.23 | 1.12 |
| Tracking S45 | | 13.47 | -0.83 | 1.13 |
| Beating S25 | 14.30 | 21.07 | 6.77 | 1.10 |
| Beating S35 | | 21.07 | 6.77 | 1.11 |
| Beating S45 | | 22.50 | 8.20 | 1.11 |
| Beating S10 | | 23.82 | 9.52 | 1.10 |


**Figure 34. Cumulative returns of the index and the three tracking portfolios in 2016**


**Figure 35. Cumulative returns of the index and the three beating portfolios in 2016**

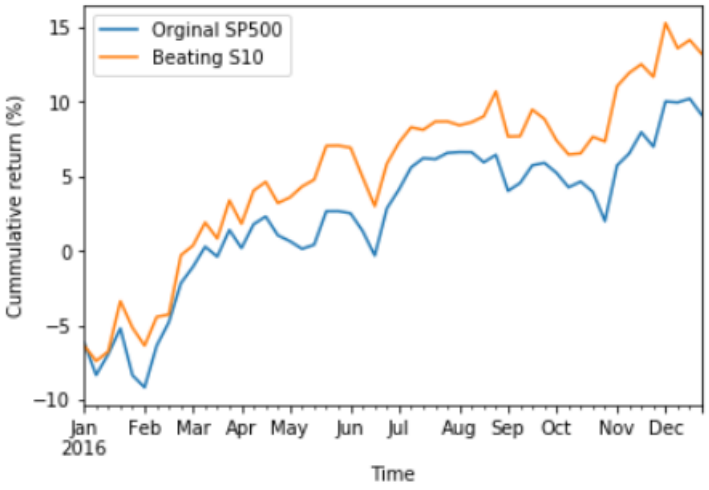


**Figure 36. Cumulative returns of the index and the beating sparse portfolio in 2016**

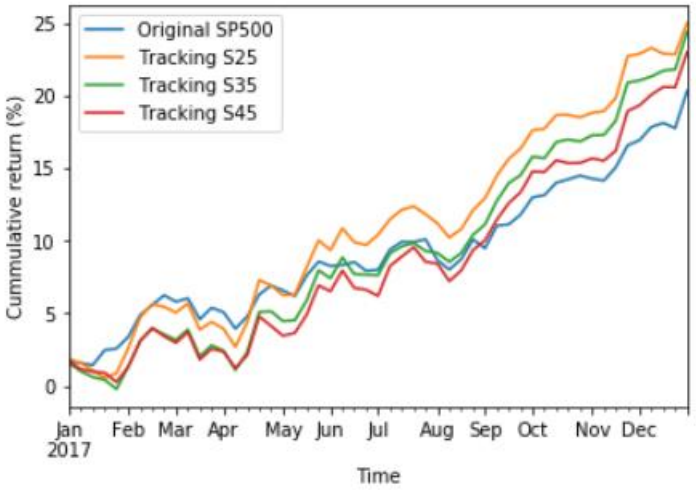


**Figure 37. Cumulative returns of the index and the three tracking portfolios in 2017**

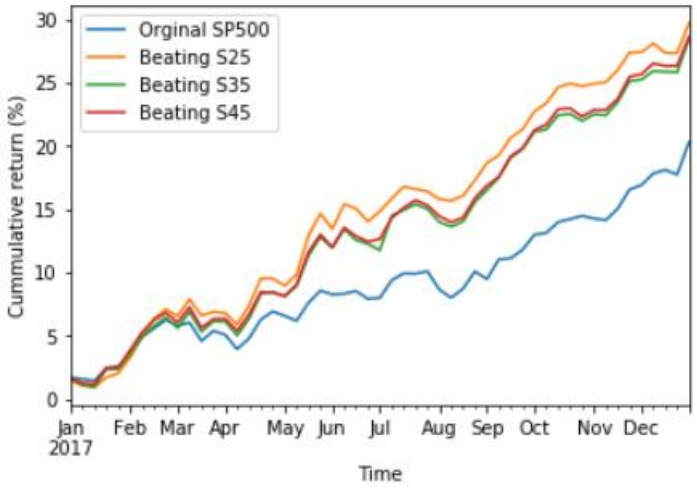


**Figure 38. Cumulative returns of the index and the three beating portfolios in 2017**

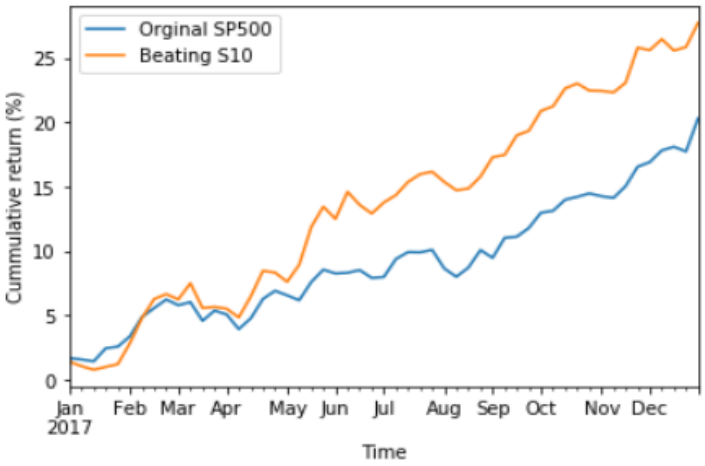


**Figure 39. Cumulative returns of the index and the beating sparse portfolio in 2017**

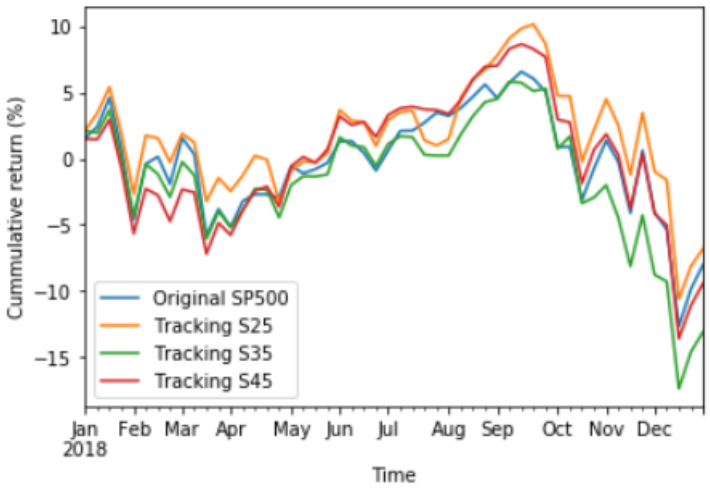


**Figure 40. Cumulative returns of the index and the three tracking portfolios in 2018**


**Figure 41. Cumulative returns of the index and the three beating portfolios in 2018**

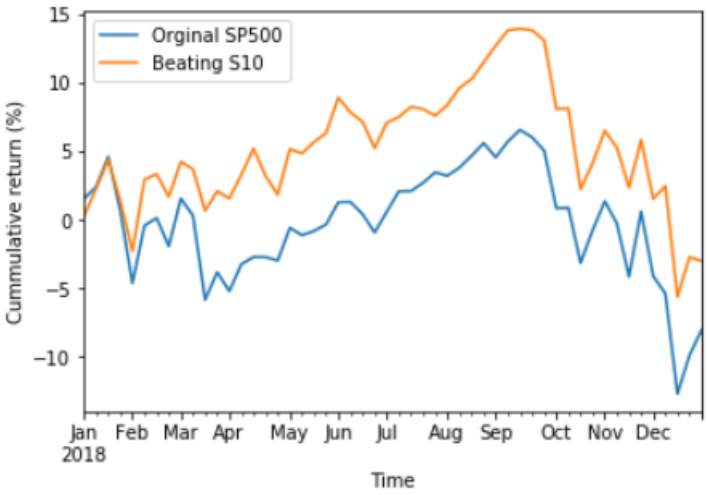


**Figure 42. Cumulative returns of the index and the beating sparse portfolio in 2018**

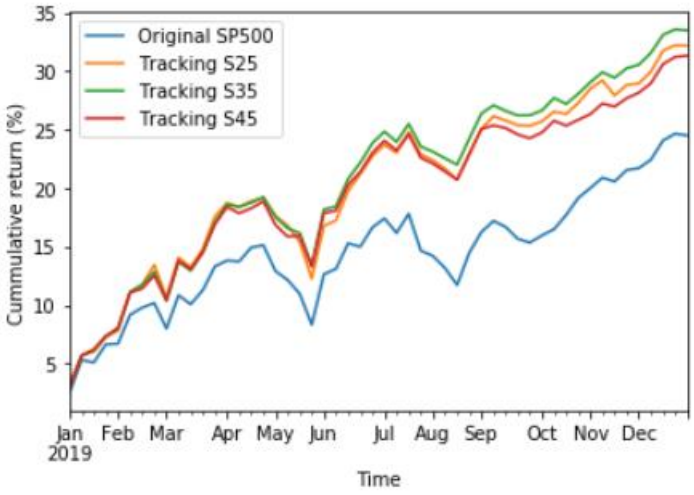


**Figure 43. Cumulative returns of the index and the three tracking portfolios in 2019**


**Figure 44. Cumulative returns of the index and the three beating portfolios in 2019**

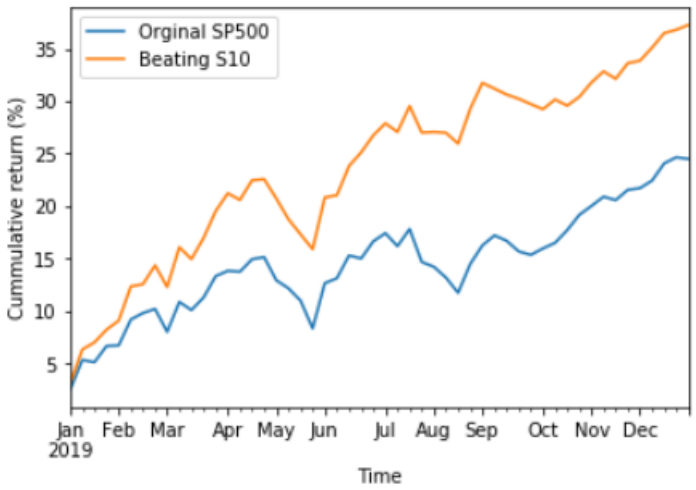


**Figure 45. Cumulative returns of the index and the beating sparse portfolio in 2019**

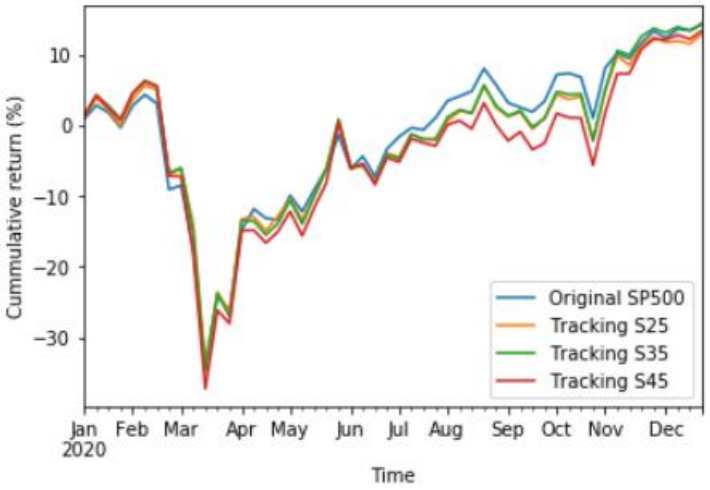


**Figure 46. Cumulative returns of the index and the three tracking portfolios in 2020**

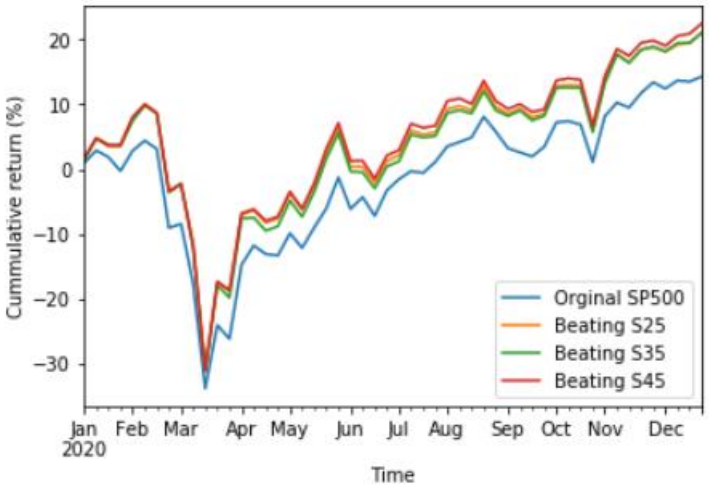


**Figure 47. Cumulative returns of the index and the three beating portfolios in 2020**


**Figure 48. Cumulative returns of the index and the beating sparse portfolio in 2020**

# 6 CONCLUSIONS

This thesis proposes a framework to construct a sparse portfolio with ten stocks beating the market index by implementing a two-step approach. Inspired by Heaton et al.'s (2017b) success, the thesis used the autoencoder model for stock selection; however, our designed networks are deeper than Heaton et al.'s. For stock weighting, the invested weights of selected stock components were determined by solving the quadratic programming problem; the thesis additionally applied $L_2$ regularization method to enhance the out-of-sample performance of the portfolios. A beating portfolio constructed under the sampling approach should both follow the trend of the market index and produce excess returns over the market index. As such, the thesis first constructed the tracking portfolio that can follow the market trend. After that, the thesis enhanced the tracking portfolios' performance by modifying the stock components' weights of the tracking portfolio. To affirm the out-of-sample performance of the constructed portfolios, the thesis arranged training and test periods continuously to obtain five different annual performances.

Deep autoencoder provides us the ranking of stock information based on its reconstruction error. The stocks that share the most common information with the market index would obtain small construction errors (the most communal stocks). In contrast, stocks obtaining large construction errors would share less common information with the market (the least communal stocks).

Therefore, the thesis constructed three tracking portfolios containing both the most and the least communal stocks to mimic the S&P500 index (joint tracking portfolio). The three tracking portfolios contain 25, 35 and 45 stocks, respectively. The empirical results show that all three portfolios can follow the market trends in all five prediction years. However, the tracking portfolios tend to underperform the index in the large drawdown period.

After examining that the selected stocks in the tracking portfolio can follow the market trends, the thesis next constructed the three portfolios beating the index with the same stocks as tracking portfolios (joint beating portfolios). The thesis then used index returns added 2% in model training to look for stocks' weights. The empirical results

show that all three joint beating portfolios can consistently beat the index in any given 1-year period even in the large drawdown in terms of cumulative abnormal returns. When taking the risk measure beta into consideration, all three joint beating portfolios have an acceptable risk level, which beta values below and close to one. However, the joint beating portfolios can be a risky investment in the large drawdown period with the beta values around 1.1.

Although the three portfolios containing a different number of stocks, the portfolio with a smaller number of stocks does not perform less well than the one with a larger number of stocks. Moreover, the most communal stocks account for most of the invested weights' proportion while invested weights of the least communal stocks are insignificant. These observations indicate that when a certain number of stocks included in the portfolio contains enough information to follow the market trends, including more stocks in the portfolio is irrelevant.

Such indication induced the thesis to construct a sparse portfolio containing only the ten most communal stocks. Despite only containing only ten stocks, the sparse portfolio does not perform less well than joint beating portfolios containing 25, 35 and 45 stocks in terms of both excess returns and riskiness in all five prediction years. Like the joint beating portfolios, the sparse portfolio is an inconsiderable-risk investment with beta values normally below and close to one (except for the beta value of 1.1 in the large drawdown period).

Thus, the thesis can answer the research question that the application of deep learning can construct a sparse portfolio beating the market index in any given 1-year period with justifiable riskiness. The thesis supports Heaton et al.'s study that the deep learning framework can solve the problem in indexing. Heaton et al. claimed that the portfolio with a larger number of stocks would obtain better performance. Their portfolio needs to contain at least 40 stocks to have a reliable prediction. However, our study showed that the portfolio with a smaller number of stocks does not perform less well than the one with a larger number of stocks. This difference may indicate that our deeper autoencoder provides better capability in selecting the representative subset of stocks.

This result brings competitive advantages for passive fund managers in many ways. First, the sparse portfolio is more economical than actively managed funds in terms of management cost as the sparse portfolio is not required to rebalance at least in a 1-year period. Second, the sparse portfolio contains only ten stocks which can save the rebalancing and investment expenses compared to the full replication approach. Third, the sparse portfolio is more beneficial than the normal tracking portfolio by generating excess returns over the market index. Particularly, after deducting management cost, the sparse portfolio still outperforms the market index while the tracking portfolio tends to underperform the market index. The framework of sparse portfolio construction not only benefits the fund managers but also offers individual investors the accessible method to construct their own portfolio beating the index with only ten stocks. However, the sparse portfolio can be risky in the large drawdown period; investors should make judicious decisions on the sparse portfolio's investment.

The study examined the performance of the sparse portfolio in a 1-year period. Further study can investigate the out-of-sample performance of the sparse portfolio in a longer horizon as well as the rebalancing or other improving strategies when the sparse portfolio starts experiencing a disappointing performance. Controlling the riskiness of the portfolio to obtain the beta value below one in the large drawdown period is also another direction for future study.

## REFERENCES


Alexander, C., & Dimitriu, A. (2004a). A comparison of cointegration and tracking error models for mutual funds and hedge funds. *ISMA Centre Discussion Papers in Finance*, *4*, 1-26. Retrieved from https://core.ac.uk/download/pdf/6565375.pdf

Alexander, C. & Dimitriu, A. (2004b). Equity indexing: Optimize your passive investments. *Quantitative Finance*, 4, (3), pp. C30-C33. https://doi.org/10.1088/1469-7688/4/3/F01

Beasley, J. E., Meade, N. & Chang, T. -J. (2003). An evolutionary heuristic for the index tracking problem. *European Journal of Operational Research*. Elsevier, vol. 148(3), pages 621-643, August. https://doi.org/10.1016/S0377-2217(02)00425-3

Benidis, K., Feng, Y., & Palomar, D. P. (2017). Sparse portfolios for high dimensional financial index tracking. *IEEE Transactions on signal processing*, *66*(1), 155-170. https://doi.org/10.1109/TSP.2017.2762286

Benidis, K., Feng, Y., & Palomar, D. P. (2018). Optimization methods for financial index tracking: From theory to practice. *Foundations and Trends® in Optimization*, *3*(3), 171-279. http://dx.doi.org/10.1561/2400000021

Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems, *Math. Program*. 74 (1996), 121–140. https://doi.org/10.1007/BF02592208

Brodie, J., Daubechies, I., De Mol, C., Giannone, D., & Loris, I. (2009). Sparse and stable Markowitz portfolios. *Proceedings of the National Academy of Sciences*, *106*(30), 12267-12272. https://doi.org/10.1073/pnas.0904287106

Cesarone, F., Scozzari, A., & Tardella, F. (2011). Portfolio selection problems in practice: a comparison between linear and quadratic optimization models. *arXiv preprint arXiv:1105.3594*. Retrieved from https://arxiv.org/ct?url=https%3A%2F%2Fdx.doi.org%2F10.1007%2Fs10287-014-0210-1&v=902f3e79

Chang, T.J., Meade, N., Beasley, J.E. & Sharaiha, Y.M. (2000). Heuristics for cardinality constrained portfolio optimisation. *Computers & Operations Research* 27(13):1271-1302. https://doi.org/10.1016/S0305-0548(99)00074-X

Chopra, V. K., & Ziemba, W. T. (2013). The effect of errors in means, variances, and covariances on optimal portfolio choice. In *Handbook of the fundamentals of financial decision making: Part I* (pp. 365-373). https://doi.org/10.1142/9789814417358_0021

Coleman, T. F., Li, Y., & Henniger, J. (2006). Minimizing tracking error while restricting the number of assets. *Journal of Risk*, *8*(4), 33. https://doi.org/10.21314/JOR.2006.134

Corielli, F. & Marcellino, M. (2006). Factor based index tracking. *Journal of Banking & Finance*, 30, (8), pp.2215-2233. https://doi.org/10.1016/j.jbankfin.2005.07.012

Derigs, U., & Nickel, N. H. (2004). On a local-search heuristic for a class of tracking error minimization problems in portfolio management. *Annals of Operations Research*, *131*(1), 45-77. https://doi.org/10.1023/B:ANOR.0000039512.98833.5a

Dunis, C.L. & Ho, R. (2005). Cointegration portfolios of European equities for index tracking and market neutral strategies. *Journal of Asset Management*, 6, (1), pp. 33-52. https://doi.org/10.1057/palgrave.jam.2240164

Fastrich, B., Paterlini, S., & Winker, P. (2014). Cardinality versus q-norm constraints for index tracking. *Quantitative Finance*, *14*(11), 2019-2032. https://doi.org/10.1080/14697688.2012.691986

Focardi, S.M., Fabozzi, F.J., 2004. A methodology for index tracking based on time series clustering. *Quantitative Finance*, 4, (4), pp. 417-425. https://doi.org/10.1080/14697680400008668

Gaivoronski, A.A., Krylov, S. & Van Der Wijst, N. (2005). Optimal portfolio selection and dynamic benchmark tracking. *European Journal of Operational Research*, 163, (1), pp. 115-131. https://doi.org/10.1016/j.ejor.2003.12.001

Gilli, M., & Këllezi, E. (2002). The threshold accepting heuristic for index tracking. In *Financial engineering, e-commerce and supply chain* (pp. 1-18). Springer, Boston, MA. https://doi.org/10.1007/978-1-4757-5226-7_1

Goodfellow I., Bengio, Y. & and Courville A.(2016). *Deep Learning.* MIT Press.

Heaton, J. B., Polson, N., & Witte, J. H. (2017a). Why indexing works. *Applied Stochastic Models in Business and Industry*. Vol 33, 690–693. https://doi.org/10.1002/asmb.2271

Heaton, J. B., Polson, N., & Witte, J. H. (2017b). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*. John Wiley & Sons, vol. 33(1), pages 3-12, January. https://doi.org/10.1002/asmb.2209

Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507. 509, 524, 528, 529, 534. https://doi.org/10.1126/science.1127647

Hoerl, A., & Kennard, R. (1988). Ridge regression, in 'encyclopedia of statistical sciences', vol. 8. *Wiley*, New York. https://doi.org/10.1002/0471667196.ess2280

Jeurissen, R, van den Berg, J. (2008). Optimized index tracking using a hybrid genetic algorithm. *In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence*). Hong Kong. https://doi.org/10.1109/CEC.2008.4631108

Karlow, D. (2012). Comparison and Development of Methods for Index Tracking. Frankfurt School of Finance & Management. Retrieved from https://d-nb.info/1054242275/34

Larsen, G. A., & Resnick, B. G. (1998). Empirical insights on indexing: How capitalization, stratification and weighting can affect tracking error. *Journal of Portfolio Management*, *25*(1), 51. https://doi.org/10.3905/jpm.1998.409656

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Li, D., Sun, X. & Wang, J. (2006). Optimal lot solution to cardinality constrained mean variance formulation for portfolio selection, *Math. Finance* 16 (2006), 83–101. https://doi.org/10.1111/j.1467-9965.2006.00262.x

Liu B. (2019). *SPIVA U.S. Scorecard*. S&P Dow Jones Indices LLC. Retrieved from https://www.spglobal.com/spdji/en/documents/spiva/spiva-us-mid-year-2019.pdf

Maginn, J. L., Tuttle, D. L., McLeavey, D. W., & Pinto, J. E. (Eds.). (2007). *Managing investment portfolios: a dynamic process* (Vol. 3). John Wiley & Sons.

Markowitz, H.M. (1952). Portfolio Selection. *Journal of Finance*, 7, 77-91. https://doi.org/10.1111/j.1540-6261.1952.tb01525.x

Maurer, F., & Williams, S. O. (2015). Physically versus Synthetically Replicated Trackers: Is There a Difference in Terms of Risk?. *Journal of Applied Business Research (JABR)*, *31*(1), 131-146. https://doi.org/10.19030/jabr.v31i1.8996

Meade, N., & Salkin, G. R. (1990). Developing and maintaining an equity index fund. *Journal of the Operational Research Society*, *41*(7), 599-607. https://doi.org/10.1057/jors.1990.84

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Montfort, K., Visser, E. & van Draat, L.F. (2008). Index tracking by means of optimized sampling. *The Journal of Portfolio Management*, 34, (2), pp. 143-152. https://doi.org/10.3905/jpm.2008.701625

Rafaely, B. & Bennell, J.A. (2006). Optimisation of FTSE 100 tracker funds: A comparison of genetic algorithms and quadratic programming. *Managerial Finance*, 32, (6), pp. 477-492. https://doi.org/10.1108/03074350610666210

Rey, D.M. & Seiler, D. (2001). Indexation and tracking errors. *WWZ, Department of Finance*, *Working Paper*, (2/01). Retrieved from https://www.econbiz.de/archiv1/2009/95490_indexation_trackingerrors.pdf

Roll, R. (1992). A mean/variance analysis of tracking error. *The Journal of Portfolio Management*, 18, (4), pp. 13-22. https://doi.org/10.3905/jpm.1992.701922

Rompotis, G. G. (2009). Active vs. passive management: New evidence from exchange traded funds. *Passive Management: New Evidence from Exchange Traded Funds (February 4, 2009)*. http://dx.doi.org/10.2139/ssrn.1337708

Rudolf, M., Wolter, H.J. & Zimmermann, H. (1999). A linear model for tracking error minimization. *Journal of Banking & Finance,* 23, (1), pp. 85-103. https://doi.org/10.1016/S0378-4266(98)00076-4

Ruiz-Torrubiano, R., & Suárez, A. (2009). A hybrid optimization approach to index tracking. *Annals of Operations Research*, *166*(1), 57-71. https://doi.org/10.1007/s10479-008-0404-4

Sharpe, W. F. (1963). A simplified model for portfolio analysis. *Management science*, 9(2), 277-293. https://doi.org/10.1287/mnsc.9.2.277

Sorensen, E.H., Miller, K.L. & Ooi, C.K. (2000). The decision tree approach to stock selection. *The Journal of Portfolio Management*, 27, (1), pp. 42-52. https://doi.org/10.3905/jpm.2000.319781

Takeda, A., Niranjan, M., Gotoh, J. Y., & Kawahara, Y. (2013). Simultaneous pursuit of out-of-sample performance and sparsity in index tracking portfolios. *Computational Management Science*, *10*(1), 21-49. https://doi.org/10.1007/s10287-012-0158-y

Troiano, L., Bhandari, A. & Villa, E.M. (2020). *Hands-On Deep Learning for Finance*. Packt Publishing Ltd. Livery Pl

Wu, L., Yang, Y., & Liu, H. (2014). Nonnegative-lasso and application in index tracking. *Computational Statistics & Data Analysis*, *70*, 116-126. https://doi.org/10.1016/j.csda.2013.08.012