Applied Project

MSc Financial Technology

2021 - 2022 Academic Year

Portfolio Selection using Deep Learning

Imperial College London

Business School

(CID: 01113462)

August 17, 2022

Client Specification

This report is based on a fictitious client CC asset management company, located in United State. The company is seeking a more efficient portfolio selection method to outperform the benchmark. In a dynamically changing market, quick decision-making and immediate reaction to opportunities are critical. However, the traditional portfolio selection strategy may be less efficient. Additionally, dealing with large quantities of data in the financial market is costly and time-consuming.

The client realized that utilizing machine learning in asset management may increase the efficiency and accuracy of investment portfolio selection. The client wishes to investigate a deep learning model which can analyze the financial dataset, identify complex patterns and insight into data, and further investigate whether the automated portfolio selection using deep learning can outperform the benchmark.

Therefore, as portfolio managers, we are hired to:

1. Build a model to analyze financial datasets which can identify complex patterns and insight into data.
2. Forecast the trend of the stock market and build a portfolio using deep learning which can outperform the benchmark, i.e., S&P 500 index.

Contents

1 Introduction **1**
1.1 Literature Review 2

2 Methodology **3**
2.1 Traditional Financial Models 3
2.2 Deep Learning 5
2.2.1 Autoencoder 5
2.3 Overall Process 6

3 Empirical Application **7**
3.1 Data Handling 7
3.2 Replicate the market behavior using Autoencoder 8
3.3 Build the portfolio 9

4 Conclusion and Discussion **10**

References **11**

A Appendices **12**
A.1 Load Data 12
A.2 Clean Data 12
A.3 Autoencoder 13
A.4 Rank the tickers based on communals 14
A.5 Build the Portfolio 14

List of Figures

1 Replicated Stocks based on communal 8
2 Highly communal 9
3 Least communal 9
4 Replicated Stocks based on communal 10

List of Tables

1 Introduction

Portfolio management is indispensable in the financial market which aims to seek the best asset allocations so that both the returns and the risk of the investment are optimized. Over the decades, it continuously attracted numerous researchers to investigate this topic, and among all the approaches, the theory of mean-variance trade-off has prevailed for more than half a century. Based on this theory, dozens of famous methods were proposed by utilizing statistical analysis. Despite the success of some models, it is extremely challenging to model the investment returns precisely by using traditional approaches, as the financial market varies dramatically over time. Traditional methods require restricted assumptions and certain prior information. However, some assumptions are not accurate for the real market, and some information is not accessible to the public. The impacts of the past on the future and the influences between the assets can be nonlinear and unforeseeable, which aggravates the difficulty of analysis. Additionally, it is also not straightforward to make a selection within thousands of assets. It almost takes non-deterministic polynomial (NP) time to compute via the traditional measures.

Fortunately, in the era of the machine learning explosion, it is possible to obtain the complex underlying non-linearity within the assets by employing deep neural networks. During the past few decades, machine learning is widely used in stock market forecasting, credit risk assessment, and portfolio allocation, and it outperforms some classical models (Ozbayo- glu, Gudelek Sezer 2020). In this paper, we introduce a deep learning approach to construct a portfolio and generate returns. We adopt deep neural networks to seek the underlying structure and interaction of financial data, create new models and generate alpha signals. The numerical results reveal that our model outperforms S & P 500 Index.

The remaining sections of the paper can be summarized as follows. The literature related to the project is described in the following part. Section 2 explains the methodology of the project, starting from the traditional portfolio construction strategy to the deep learning framework, and then discusses the main idea of the project. In section 4 Empirical Application, the data selection process and the results of the model simulation will be presented. The last section provides the conclusion, discussion, and additional future work related to the project.

1.1 Literature Review

Markowitz (1952) proposed the mean-variance portfolio construction approach in his paper Portfolio Selection in 1952 known as Modern Portfolio Theory (MPT). The traditional mean-variance optimal (MVO) is an optimization method that pursues the maximum return portfolio under a given risk or the minimum risk portfolio under a given return. Fischer Black and Robert Litterman proposed the Black-Litterman model in 1991. Black & Litterman (1991) combines the mean-variance portfolio construction theory proposed by Markowitz (1952), the CAPM theory proposed by Sharpe (1964), and the hybrid estimation method proposed by Thiele & Chan (1978) in the model. The Black-Litterman model uses the method of Bayesian statistics, takes the market equilibrium rate of return as a prior distribution, and combines the subjective views of investors to finally obtain an estimate of the expected rate of return. By introducing this expected rate of return in the mean-variance portfolio theory, the asset allocation strategy guided by investors' views can be obtained.

Portfolio management and selection models based on deep learning started getting more attention in both academia and industry. Takeuchi & Lee (2013) analyze the stocks from NYSE, AMEX, and NASDAQ using autoencoder and RBM, and then sort the stocks into high momentum and low momentum based on return rate, then build the portfolio and achieved a high return. Aggarwal & Aggarwal (2017) use deep learning (single and multi neural network) to predict whether their selection of stocks outperforms or underperforms the benchmark index. By using the predicted outcome, they adjust the weight of stocks in the portfolio and finally outperform the target.

In general, portfolio management can be categorized into two approaches, passive management (replicate the asset return on benchmark index) and active management (beat the market). Frino & Gallagher (2001) investigated and compared the performance between passive index funds and active mutual funds, and concluded that S & P 500 index funds outperformed active funds. Further, Wu et al. (2007) indicated that suitable trade-offs between alpha generation and tracking error enhanced the index strategy. Grace (2017) uses auto-encoder and LSTM to predict the stock price, and constructed a smart index based on prediction.

Our work is based on the deep portfolio theory which was first published by J. B. Heaton, N. G. Polson, and J. H. Witte. In their paper, Deep Learning for Finance: Deep Portfolio (Heaton, Polson & Witte 2017), they used the deep

learning method to replicate the financial data and build the portfolio. To be more specific, an autoencoder is employed to analyze the potential interactions of data for the stock of the biotechnology IBB index. They find the nonlinearity relationship of data, compute the component weights and construct an index fund that outperforms the target index.

2 Methodology

2.1 Traditional Financial Models

Before diving into the deep learning implementations, it would be beneficial to mention the traditional portfolio selection model. Markowitz's modern asset allocation theory takes the expected returns and covariance matrices of these investments as inputs, and finds the best portfolio by optimizing the following objective function

$$\max_{w} \; w'\mu - \frac{1}{2}\sigma w'\Sigma w. \tag{1}$$

If there is no constraint, we can get

$$w \;=\; (\sigma\Sigma^{-1})\mu, \tag{2}$$

where
μ denotes the expected return vector of the investment product,
Σ stands for the covariance matrix of the investment product,
σ represents the investor's risk aversion coefficient,
w expresses the allocation weight of the investment product in the portfolio.

Compared with the Mean-Variance model, the Black-Litterman model takes the market equilibrium excess return implied by CAPM as the prior distribution and combines the subjective views of investors (new information) to finally obtain the posterior distribution of the expected rate of return.

The prior distribution can be written as

$$P(A) \sim N(\Pi, \tau\Sigma), \tag{3}$$

where Π denotes the vector of equilibrium excess returns for each asset and τ represents the constant of proportionality. The new information can be formulated as

$$P(B|A) \sim N(P^{-1}Q, [P^t\Omega^{-1}P]^{-1}), \tag{4}$$

where
P denotes a $k \times n$ matrix of the asset weights within each view.
Q expresses a $k \times 1$ vector of the returns for each view.
Ω stands for a $k \times k$ matrix of the covariance of the views.
Consider Bayes Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{5}$$

To maximize the probability $P(A|B)$, we can solve the objective function

$$\min_{\mu} (\mu - \Pi)'\tau\Sigma(\mu - \Pi) + (P\mu - Q)'\Omega(P\mu - Q), \tag{6}$$

or, equivalently

$$\min_{\mu} \ \|\mu - \Pi\|^2_{\Sigma} + \lambda\|P\mu - Q\|^2_{\Omega}, \tag{7}$$

where $\lambda = \frac{1}{\tau}$. Finally we can obtain the posterior distribution

$$P(A|B) \sim N([(\tau\Sigma)^{-1}\Pi + P^t\Omega^{-1}Q][(\tau\Sigma)^{-1} + P^t\Omega^{-1}P]^{-1}, (\tau\Sigma)^{-1} + [P^t\Omega^{-1}P]^{-1}). \tag{8}$$

When using mean-variance theory for asset allocation, as long as the return and covariance matrix of the next period of assets are given, the optimal investment portfolio can be calculated. The allocation result of mean-variance will long the assets with good historical performance and short the assets with poor historical performance. But the historical return rate of the assets basically cannot predict the future, so this kind of optimization generally will perform poorly in the future. In conclusion, the expected returns are too difficult to predict. No one can accurately predict the returns of all assets in the next period. The optimization results of Markowitz's modern asset allocation are very sensitive to the input. Little input changes to the expected return will cause large fluctuations in the results.

The Black-Litterman asset allocation model addresses those problems in the Markowitz model. When there is no new information, since the market is efficient, we can assume the next period's rate of return is the market equilibrium rate of return. In addition, In the Black-Litterman model, the view of experi-

enced investors is considered new information. Transforming this information into next-period returns through Bayesian statistical techniques, rather than subjective adjustments, is a more reliable method to incorporate views into asset allocations. However, the limitation of the Black-Litterman model is the accuracy of the model highly depends on accurate views provided by investors. If the investor provides a less proficient view of the investment, then the outcome of portfolio selection may perform poorly than expected. In addition, the nonlinear features like jumps and volatility in history data are important while the traditional model normally ignores them.

2.2 Deep Learning

A shortcoming of the current state of the art for portfolio selection model requires accurate information from the history dataset to achieve good accuracy. Meanwhile, the deep learning model can analyze the underlying nonlinear relationship no matter how complex the data is. Therefore, deep learning is introduced in portfolio selection to analyze the latent information of the data and minimize the error.

2.2.1 Autoencoder

An autoencoder is an unsupervised neural network model, which consists of two parts called encoder and decoder. It is trained to replicate the input x into output $\tilde{x}$ approximately by minimizing the error between them. Autoencoder learns the potential features of the input data in the encoder and reconstructs the original input data with the learned new features in the decoder. Normally, an autoencoder can be used in dimension reduction and a feature extractor. The details of the autoencoder can be explained as follows.

Input:

$$x = [x_1, x_2, x_3 ... x_{dx}]', \tag{9}$$

where dx is the dimension of input x.

Encoder: Converting x from the input layer to the hidden layer through the encoding mapping function f(x), and get $h = [h_1, h_2, h_3 ... h_{dh}]$

$$h = f(x) = \sigma(Wx + b), \tag{10}$$

where dh is the dimension of the hidden layer, w is the $dhxdx$ weight matrix and b is the bias vector. The activation function can be a sigmoid function, a tanh function or a Relu function.

Decoder: Mapping h to $\tilde{x}$ through decoding mapping function g(x),

$$\tilde{x} = g(h) = \sigma(\tilde{W}h + \tilde{b}), \tag{11}$$

where w is the $dhxdx$ weight matrix and b is the bias vector. The activation function σ can be a sigmoid function, a tanh function, or a Relu function. Output:

$$\tilde{x} = [\tilde{[}x_1], \tilde{[}x_2], ...\tilde{[}x_{dx}]] = [g(f(x1)), g(f(x2)), ...g(f(x_{dx}))]. \tag{12}$$

The learning process can be simply described as a loss function

$$L(x, g(f(x))). \tag{13}$$

Regularization is a process of adding a penalty term according to the prior information of the coefficients to obtain a sensible solution and avoid overfitting. Regularized autoencoders utilize a loss function that promotes the model to have additional features outside the ability to replicate its input to its output, instead of restricting the potential of the model by keeping the encoder and decoder shallow and the code size minimal (Goodfellow, Bengio & Courville 2016). Therefore, the loss function can be reconstructed as follows, where $\Omega(w)$ is a penalty term:

$$L(x, g(f(x))) + \lambda\Omega(w), \tag{14}$$

where λ is a penalty parameter for regularization.

We introduced the Ridge Regression in the model; it estimates parameters by minimizing the following objective function:

$$L(x, g(f(x))) + \lambda\|w^2\|_2^2. \tag{15}$$

2.3 Overall Process

We proposed a data driven model to investigate the nonlinearity relationship of the data and build the portfolio. The whole process can be divided into four steps. The first phase is to load the raw data and clean the data. Before feeding

the cleaned data into the deep learning model, we should scale the data using normalization. Then, we build the first autoencoder model to replicate the input itself. The first autoencoder may extract the market information and obtain the market behavior. Feed the raw data stock by stock to the first autoencoder, we can obtain similarities of each stock compared to the market.

In the next phase, we shall select stocks to build the portfolio. The stocks are ranked on the basis of similarity, which indicates the correlation of every stock with the market. To build the portfolio, N most similar and M least similar stocks are selected to build the portfolio. In the last step, we build another shallow autoencoder and feed 25 stock data into the model. By training the model, we can finally have the weight of each stock that enables the portfolio to outperform the target.

3 Empirical Application

We use daily return data from the United States market, which listed 500 stocks from the beginning of 2017 to the end of 2021. Using the deep learning method, we can obtain a model that can automatically generate a portfolio that achieves the target return rate.

3.1 Data Handling

As Standard & Poor's 500 is the proxy for the stock market in the United States. In the project, we use the S & P 500 stock data as the stock market dataset. The historical stock price of the 500 companies' stock price from 2017/01/03 to 2021/12/30 is downloaded from Yahoo Finance, which includes all related information, such as the open price, close price, and trading volume. The study covers the calendar years 2017 to 2021.

Stocks with incomplete data should be discarded. The first step of data processing is to remove the unnecessary characters and delete the NAN data. Then we calculate the daily log return of each stock. To make the comparison meaningful, we unify data to the same scale of magnitude by normalizing the data.

3.2 Replicate the market behavior using Autoencoder

We need to map the market structure by analyzing historical stock return data. The autoencoder is a deep learning method that replicates itself based on income x, which is normally used to extract latent information from the dataset. We use the autoencoder to explore undiscovered information that cannot be found by the traditional portfolio selection model.

The model is the autoencoder with 500 neurons and 4 layers, fed with all clean data (total stock log return from 500 companies). The learning rate, which is a hyperparameter used to update weights during gradient descent, is set to 0.001. The lower the learning rate, the slower the changes in the loss function. Using a lower learning rate enables us to find local minima but takes us longer to converge and may cause overfitting. However, if the learning rate is too high, it is prone to gradient explosion, and the model is difficult to converge. Thus, we use a hyperparameter selection model to select an optimal value of the learning rate. The number of epochs is set to 500 and the batch size is 64. The loss function used in the model is the mean square error (MSE).

Through the training process, the model replicates the behavior of the market as the model learned each stock's performance in the same period. Then we train the model with each company's log return as the input to get the copied log return. By comparing the copied log return and the original log return, the similarity of the stock can be obtained. The stocks are ranked based on the similarity to the replicated log return trained by the model. The similarity is represented by the MSE. The figure below illustrates the ranked stocks.

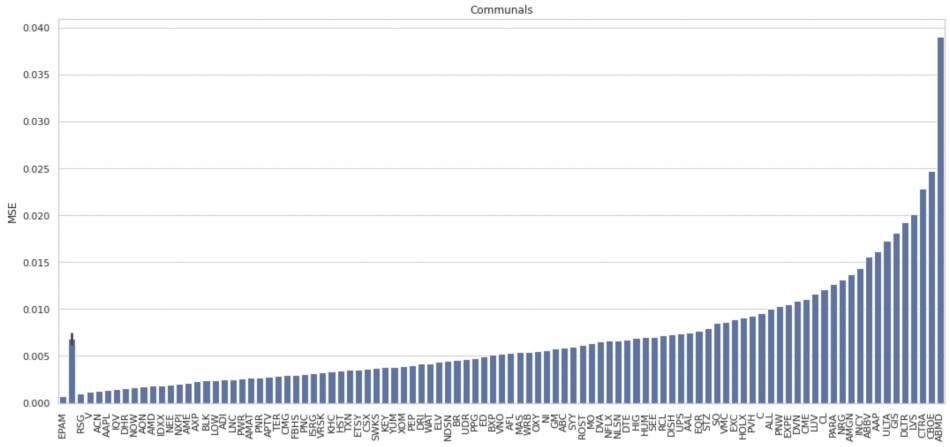

Figure 1: Replicated Stocks based on communal

The stock with the highest similarity (lowest MSE) with the replicated result

is called the most communal, therefore it is highly correlated to the market. The stock with the least similarity (Highest MSE) with the replicated result is called the least communal and therefore is less correlated to the market.

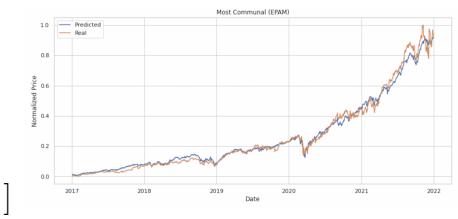

Figure 2: Highly communal

Figure 3: Least communal

3.3 Build the portfolio

Markowitz (1952) indicated that when constructing a diversified portfolio the correlation between the assets should be considered. The higher the correlation between returns on assets, the higher the risk it brings to the portfolio. Therefore, to diversify the portfolio, we select the top 10 most communal stocks and the 15 least communal stocks to construct the portfolio. The least communal stocks are considered more information efficient since they are less correlated with autoencoder or the stock market.

Totally 25 stocks are selected to build the portfolio, from which 10 stocks are selected from the most communal stock and 15 of them selected from the least communal stocks. In the second phase, we need to train an autoencoder to obtain the weight of each stock in the portfolio which can outperform the target.

The model is the autoencoder with 25 neurons and 2 layers. The activation function is tanh, so we can constrain the weight within [-1,1]. Therefore, in our model, we can both long and short the stock. The hyperparameters are set as follows: learning rate(0.001), number of epochs (500), batch size(32) and loss function is MSE.

The training result is shown in Figure 5. The benchmark is S&P 500 index. We set the target return rate to be 5% higher than the benchmark. The prediction result is the return of the deep portfolio. Our portfolio tracks the index with a deep learning method. From the result, we can see that the portfolio constructed by the autoencoder can outperform the S & P 500 index. It should be noted that in 2020, although there is a dramatic decline caused by the coronavirus pandemic,

the portfolio still obtains a higher return than the real index. It shows that the model is anticorrelated in periods of large drawdowns.

Figure 4: Replicated Stocks based on communal

4 Conclusion and Discussion

A new approach to portfolio selection using deep learning which can enhance index investing is introduced in the paper. We formulate the portfolio construction as an encoder-decoder process using an autoencoder. Moreover, the constructed portfolio outperforms the benchmark. Utilizing deep learning in portfolio selection can generate a return that has a huge potential to investigate further.

The next steps in the future will be to investigate a more efficient deep learning method. It would be interesting to adapt the autoencoder to extract underlying data involving macroeconomic characteristics.

References

Aggarwal, S. & Aggarwal, S. (2017), 'Deep investment in financial markets using deep learning models', *International Journal of Computer Applications* **162**(2), 40–43.
URL: *http://www.ijcaonline.org/archives/volume162/number2/27218-2017913283*

Black, F. & Litterman, R. (1991), 'Allocation with equities, bonds and currencies', *Goldman Sachs and Co.* .

Frino, A. & Gallagher, D. R. (2001), 'Tracking s&p 500 index funds', *The Journal of Portfolio Management* **28**(1), 44–55.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

Grace, A. (2017), 'Can deep learning techniques improve the risk adjusted returns from enhanced indexing investment strategies'.

Heaton, J. B., Polson, N. G. & Witte, J. H. (2017), 'Deep learning for finance: deep portfolios', *Applied Stochastic Models in Business and Industry* **33**(1), 3–12.

Markowitz, H. (1952), 'Portfolio selection', *Journal of Finance* **162**, 77–9.

Sharpe, W. (1964), 'A theory of market equilibrium under conditions of risk', *Journal of Finance* **19**(3), 425–444.

Takeuchi, L. & Lee, Y.-Y. A. (2013), Applying deep learning to enhance momentum trading strategies in stocks, *in* 'Technical Report', Stanford University Stanford, CA, USA.

Thiele, G. & Chan, G. (1978), 'Application of the hybrid technique to time domain problems', *IEEE Transactions on Antennas and Propagation* **26**(1), 151–155.

Wu, L.-C., Chou, S.-C., Yang, C.-C. & Ong, C.-S. (2007), 'Enhanced index investing based on goal programming', *The Journal of Portfolio Management* **33**(3), 49–56.

A Appendices

A.1 Load Data

```
def loadData(Tickers,startDate,endDate):
 stock_final = pd.DataFrame()
# iterate over each symbol
 for i in Tickers:

    print( str(i) + str(' : ') + i, sep=',', end=',', flush=True)

    try:
        # download the stock price
        stock = []
        stock = yf.download(i,start=startDate, end=endDate, progress=Fal

        # append the individual stock prices
        if len(stock) == 0:
            None
        else:
            stock['Name']=i
            stock_final = stock_final.append(stock,sort=False)
    except Exception:
        None
 return(stock_final)

payload =pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_c
SP_table = payload[0]
start_date = "2017-01-01"
end_date = "2022-01-01"
symbols = SP_table['Symbol'].values.tolist()
df = loadData(symbols,start_date,end_date)
```

A.2 Clean Data

```
df1 = df.reset_index()
df1 = df1[['Date', 'Close', 'Name']]
df1 = df1.set_index(['Date','Name']).unstack('Name')
df1 = df1.Close

df1.isna().sum().sort_values()
nalist = df1.isna().sum()
df1.drop(nalist[nalist!=0].index, axis=1, inplace=True)
df1.isna().sum().sum()
plt.figure(figsize=(14,6))

dftest=df1.iloc[:,:10]
```

```
#dftest.plot(figsize=(10,6))

dftest = df1.copy()
min_max_scaler = MinMaxScaler()
df_scaled = min_max_scaler.fit_transform(dftest)
df_scaled = pd.DataFrame(df_scaled, index=dftest.index, columns=dftest.c
spy_re = np.asarray(spy).reshape(-1,1)
spy_scaled = min_max_scaler.fit_transform(spy_re)
spy_scaled = pd.DataFrame(spy_scaled, index = spy.index)
```

A.3 Autoencoder

```
input_dim = df1.shape[0]
layer_1_dim = 500
layer_2_dim = 5
output_dim = input_dim
lambd = 10e-6

regularizer = tf.keras.regularizers.L2(lambd)


autoencoder = tf.keras.models.Sequential([
    layers.Dense(layer_1_dim, input_shape=(df1.shape[0],),
                 activation='relu',
                 use_bias=True, bias_initializer='zeros',
                 activity_regularizer=regularizer),
    layers.Dropout(0.1),
    layers.Dense(layer_2_dim, activation='relu',
                 use_bias=True, bias_initializer='zeros'),
    layers.Dense(layer_1_dim, activation='relu',
                 use_bias=True, bias_initializer='zeros'),
    layers.Dropout(0.1),
    layers.Dense(output_dim, activation='tanh',
                 use_bias=True, bias_initializer='zeros'),
])

autoencoder.summary()


opt = keras.optimizers.Adam(lr=0.001)

autoencoder.compile(optimizer=opt, loss=losses.MeanSquaredError())

autoencoder.fit(df_scaled.T, df_scaled.T,
                epochs=500,
                batch_size=64)
```

A.4 Rank the tickers based on communals

```
from sklearn.metrics import mean_squared_error
mean_squared_error(df_pred, df_scaled)
mses = []

for i in range(df_scaled.shape[1]):
  mses.append(mean_squared_error(df_pred.iloc[:,i], df_scaled.iloc[:,i])

df_mse = pd.DataFrame(mses, index=df_scaled.columns)
df_mse_sorted = df_mse.sort_values(by=0)
DF_MSE_SORTED = df_mse_sorted[0]
index = DF_MSE_SORTED.index
cutted_index = [i for i in index]
for i in range(len(cutted_index)):
  if i%5 != 0 and i != 0 and i!= len(cutted_index)-1:
    cutted_index[i] = ''
for i in range(2,5):
  cutted_index[-i] = ''
MSE_df = pd.DataFrame(DF_MSE_SORTED.values, index = cutted_index, column

MSE_df_test = MSE_df[:]

plt.figure(figsize=(18,8))
#sns.set_color_codes("muted")
#sns.barplot(x=MSE_df[0].index, y =MSE_df[0].values, color='b')
sns.barplot(x=cutted_index, y=MSE_df_test['MSE'], color='b')
plt.ylabel('MSE')
plt.xticks(rotation=90)
plt.title('Communals')
plt.show()
```

A.5 Build the Portfolio

```
autoencoder2 = tf.keras.models.Sequential([
    layers.Dense(layer_2_dim, input_shape=(num_porfolio,),
                 activation='relu',
                 use_bias=True, bias_initializer='zeros',
                 activity_regularizer=regularizer)
    ,
    layers.Dense(num_porfolio, activation='tanh',
                 use_bias=True, bias_initializer='zeros'),
])

autoencoder2.summary()

opt = keras.optimizers.Adam(lr=0.001)
```

```
autoencoder2.compile(optimizer=opt, loss=losses.MeanSquaredError())

history = autoencoder2.fit(X_train, y_train,
                  epochs=200,
                  batch_size=64,
                  )

pred = autoencoder2.predict(X_train)
pred = pd.DataFrame(pred[:,0], index=df_potfolio_lrr.index)
```