Comparison of the Capital Asset Pricing Model (CAPM) and an Artificial Neural Network Approach

Sahadatu Larabu (sahada@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Prof Kanshukan Rajaratnam
Stellenbosch University, South Africa

14 May 2020

Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

The Capital Asset Pricing Model (CAPM) is an outstanding development in asset pricing theories which was introduced by William Sharpe (1964) and John Lintner (1965). It has been universally used by financial practitioners despite it being empirically shown to have limitations. In this essay, we examine the Capital Asset Pricing Model used for measuring risks and predicting stock returns on five companies stocks listed on the New York Stock Exchange (NYSE). The data used in this study pertains to Apple Inc. (AAPL), Microsoft Corporation (MSFT), Alphabet Inc. (GOOGL), Amazon, JPMorgan Chase (JPM), and the S&P500 index (GSPC) from January 1, 2000 to December 31, 2019. We compare two models. First we predict stock returns using the CAPM. Secondly, we relax the linear relationship assumption between the returns and use a Feed-forward Neural Network (FFNN) to predict stock returns. More precisely, with the Multi-layer Perceptron (MLP). In comparing the two models, the expected returns of these models are compared with the returns from the historical data of each of the company stocks using the Mean Squared Error (MSE) as the performance measure. The MSE for the MLP was less than that of the CAPM for all the stocks except for JPM. Therefore showing that MLP outperforms the CAPM in most instances. This finding indicates that the FFNN should be considered a tool in predicting returns.

Keywords: CAPM, Linear Regression, FFNN, MLP.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Sahadatu Larabu, 14 May 2020

Contents

1. Introduction

Financial academics and experts are concerned with the development of asset pricing theories and models for measuring risks and returns due to the fluctuation dynamics and volatilities in the stock market. Harry Markowitz, a renowned Nobel prize award winner for his spearhead contributions to financial economics and corporate finance established the Modern Portfolio Theory (MPT). MPT is an investment scheme for portfolio selection and development based on maximising returns and minimising risks simultaneously. Moreover, after Harry Markowitz's MPT, diverse asset pricing theories like the Arbitrage Pricing Theory (APT) and Capital Asset Pricing Model (CAPM) were introduced with the goal of explaining excess market returns and risk measure. However, some of these theories use external forces like macroeconomic factors and others use internal forces like variance and covariances. CAPM is one of the internal force types of asset pricing theories.

The CAPM was a leap introduced by William Sharpe (1964) and John Lintner (1965) (Sharpe, 1964) which begets the theories of asset pricing (Fama and French, 2003) leading to a Nobel prize in 1990 to William Sharpe. This model establishes a linear relationship between expected returns and the market risk premium of an asset. It measures expected returns by taking the summation of the risk-free rate and beta multiplied with the market risk premium indicating that an asset's risk premium is determined by its beta. Beta is the representation for the non-diversifiable or systematic risk which is a relative measure of the asset risk to that of the market risk. The CAPM assumes that beta is sufficient in explaining asset prices. Four decades after its introduction, the CAPM was still popularly used in assessing the performance of portfolios, estimating the cost of shareholder equity for industries or firms, and measuring risks (Fama and French, 2003). It has been the center of attraction for investment courses. The CAPM follows some assumptions which are unrealistic in the world. Out of these assumptions, one of them is that investors understand investment especially in calculating returns in a similar manner (Kisman and Restiyanita, 2015). We will throw more highlights on the assumptions and drawbacks of the CAPM in Section 2.1 of Chapter 2. However the CAPM theory follows a simple logic and its predictions are intuitively pleasing making it very attractive.

In this era of the data economy, machine learning is currently a field of great interest in science and technology used in various fields like finance, medicine, agriculture, and engineering. Machine learning is where computers or algorithms learn from previous data or experiences and can make predictions on unseen data or make decisions for the future without been directly programmed. Speech recognition, facial recognition, and self-driving cars would not have been possible without machine learning. Moreover, this field has become a useful mechanism for making predictions on financial data or the stock market used by financial practitioners. It can also be used in predicting and calculating asset returns like the CAPM and other asset pricing theories. We will give a brief overview of machine learning processes and types in Section 2.2 of Chapter 2.

1.1 Related Works

There have been several key critiques about the overall efficacy of the CAPM (Womack and Zhang, 2015). In some of the cases it was commended whiles it was criticised in others. Empirical studies showed the differences in returns was not captured in the betas. That is the assumption of only one factor (the market risk premium) explaining expected return was wrong. Researchers believe that other factors not captured in the CAPM have an impact on returns in the market (Womack and Zhang, 2015). There have been several asset pricing theory attempts made. Some of which are APT and

others seen in this section. The APT was introduced by Stephen Ross (1976) which uses multiple factors (macroeconomic variables) as an explanatory aspect of returns. Kisman (2015) and Restiyanita (2015) examined the CAPM's prediction with that of the APT on the Indonesia Stock Exchange and showed that the APT was better in explaining returns (Kisman and Restiyanita, 2015). In the early years of the 1990s Eugene Fama and Kenneth French proposed the three-factor model which takes into account the market risk, the market capitalization "(SMB-Small Minus Big)" and the book-to-market ratio "(HML-High Minus Low)" (Womack and Zhang, 2015). The CAPM and the three-factor model were examined for the Dhaka Stock Exchange and concluded on the three-factor model predicting preferably than the CAPM (Sattar, 2017). The three-factor model was extended in 2015 to a five-factor model by including profitability and investment factors (Fama and French, 2015). These three-factor and five-factor models were examined for the Turkish Stock Exchange and showed to outperform the CAPM in predicting returns (Erdinç, 2017). There have also been some studies comparing the CAPM, the three-factor model and Artificial Neural Networks (ANN). The forecasting ability of these models for the Shanghai Stock Exchange was compared. The ANN was found to perform better than the linear model that is the CAPM and the three-factor model (Cao et al., 2011).

1.2 Aims and Objectives

This essay aims to examine the CAPM and to overcome some of its limitations when we have more complex and non-linear relationships between the individual returns and the excess market returns using machine learning. We will adopt machine learning techniques like the Multi-layer Perceptrons (MLP) using the same explanatory variable for the CAPM but allows for data non-linearities and complexities on five companies listed on the New York Stock Exchange. The historical daily stock data for Apple Inc. (AAPL), Microsoft Corporation (MSFT), Alphabet Inc. (GOOGL), Amazon, JPMorgan Chase (JPM) and the S&P500 index (GSPC) from January 1, 2000 to December 31, 2019 will be used. The expected returns of the CAPM and the MLP are compared using the returns from the historical stock data of each of the companies and calculating the Mean Square Error (MSE) as our loss function for the performance measure for each of these company stocks.

1.3 Essay Layout

In this essay, the subsequent chapters follow this outline. We summarise past literature on the CAPM, machine learning, linear regression analysis, regularization techniques and performance measures in Chapter 2. We then discuss artificial neural networks, feedforward neural networks, how neural networks learn and the problems in training a neural network model in Chapter 3. Furthermore, Chapter 4 encompasses the experiments we carried out, discussions about the results and comparisons. We finally give a conclusion on our work in Chapter 5.

2. CAPM and Regression Analysis

In this chapter, we give a brief overview of the CAPM, the assumptions and drawbacks of the model in Section 2.1. We then give a brief overview of machine learning in Section 2.2. Next, we discuss regression analysis in Section 2.3 and then lay more emphasis on simple linear regression and least squares estimation method in Section 2.4. In Sections 2.5 and 2.6 we explain regularization techniques and the evaluation measures for regression models respectively.

2.1 A Brief Overview of the Capital Asset Pricing Model

The CAPM is an asset pricing theory originally proposed by William Sharpe (1964) and John Lintner (1965) (Sharpe, 1964) . This model establishes a linear relationship between excess returns and market risk premium of an asset. The CAPM is used in calculating the performance of portfolios, cost of shareholder equity for firms and measuring risks (Fama and French, 2003). The relationship between returns and the risk premium for the CAPM is expressed as

$$E(R) = R_f + \beta(ER_m - R_f) \tag{2.1.1}$$

where $E(R)$ is the expected returns of the asset, R_f is the risk free rate, β is the non-diversifiable or systematic risk measured using covariance, ER_m is the expected market returns. $ER_m - R_f$ is the market risk premium.

Furthermore, investors are compensated for risks and the time value of money, hence the R_f accounts for the time value of money and the $ER_m - R_f$ accounts for the risk incurred. For the beta, a beta value of one shows that the volatility of the asset is the same as that of the market average. Whiles a beta value greater than one shows the asset's volatility is greater than that of the market average and vice versa.

2.1.1 Assumptions and Drawbacks of the CAPM.

The CAPM is based on several assumptions. These assumptions are (Goetzmann et al., 2014):

- Investors do not incur transaction cost and personal income tax.
- Assets are divisible boundlessly, investors can buy any amount of investment.
- A single investor does not influence the price of stock either by buying or selling.
- Investors make investment decisions based on the expected value and the variance of the returns of their portfolio.
- Investors can sell short any amount of shares.
- Investors can borrow and lend any amount of funds at a risk-free rate.
- All investors have the same expectations considering the expected returns, variance and correlation between stock pairs in the portfolio.
- All assets can be bought or sold including human capital on the market.

Although the CAPM is easy to interpret, it has several drawbacks since the assumptions on which this model is based are unrealistic in the real world. A hand few of these drawbacks are (Diksha, n.d.)

- As seen in Section 2.1 beta is used as a measure of risk but beta is unstable and is not a good estimate of the volatility of assets (stocks).
- The CAPM focuses on systematic risk but empirical evidence has found total risks to be more relevant and also have a linear relationship with returns.

2.2 A Brief Overview of Machine Learning

Machine learning is a field of study where computer science and statistics principles are used to develop statistical models for performing tasks like predictions and conclusions (Nwankpa et al., 2018). These models are in the form of a mathematical relation between the inputs and outputs of a given system. In this section we will briefly discuss the machine learning procedures and the various types of machine learning. The purpose of machine learning is to be able to minimize the cost or error function of the mathematical model constructed from the data using various optimization techniques and make predictions on new data (Goodfellow et al., 2016)

There are three broad classes of machine learning. These are supervised learning which entails training the model using labelled datasets. Unsupervised learning which entails training the model using unlabelled data and reinforcement learning which is a reward-based learning type where feedbacks are given back to the agent or algorithm, a reward when it gets a task right and a penalty otherwise. Some examples of supervised learning are regression, random forest, and support vector machines and for unsupervised learning, there is k-means clustering.

The greater the number of observations in the data, the better the model and the better the prediction and accuracy. The inputs are called the features and the outputs are the labels or targets. The machine learning process involves:

- Collection of data and preparation
- Selection of feature
- Choice of algorithm
- Selection of the model
- Training and Evaluation

2.3 Regression Analysis

Regression analysis is a supervised statistical and machine learning technique used to analyse datasets (Frees, 2009). It analyses the relationship between one or more dependent variables (or regressands or features or explanatory variables) and independent variables (or regressors or labels). There are three types of regression analysis. These are the simple linear regression, the multiple linear regression and the non-linear regression. For the simple linear regression, we have only one feature and one label with a linear relationship between these variables. Whiles in a multiple linear regression we have two or more features and one label with a linear relationship. However, in the non-linear regression there is no linear relationship between the features and labels.

Moreover, regression is useful in predicting regressand values based on a set of values of regressors (Frees, 2009). That is given a feature x what is its corresponding label value y.

2.4 Simple Linear Regression

A simple linear regression model is a linear regression model which uses only 1 explanatory variable to predict a dependent variable. This can be related to the straight line equation given by $y = mx + c$. Lets suppose we have n sets of data pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where the x_i's are the features and at the y_i's are the labels for $i = 1, 2, .., n$ then the simple linear regression model is expressed as

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{2.4.1}$$

where the ϵ_i's are the error terms with a normal distribution with a 0 mean and a σ^2 variance and β_0 and β_1 are the regression coefficients.

In these simple linear regression models, we therefore seek to find the optimal estimates for β_0 and β_1 that best describes the data or the best-fit regression line for the data to be able to predict y for a given x.

2.4.1 Least Squares Estimation Method.

Least square estimation method is an optimisation technique use in finding an estimate for the parameters β_0 and β_1 which gives the least residual sum of squares (RSS) (Pattanayak et al., 2017), by choosing the line closest to all the data pairs (x_i, y_i) that is we want the coefficients such that the sum of the squared difference between the actual label y_i and the predicted label

$$\hat{y_i} = \hat{\beta_0} + \hat{\beta_1} x_i \tag{2.4.2}$$

over all the data pairs is minimum. The RSS is given by

$$\sum_{i=1}^{n} (y_i - \hat{y_i})^2 = \sum_{i=1}^{n} \left(y_i - (\hat{\beta_0} + \hat{\beta_1} x_i)\right)^2 \tag{2.4.3}$$

This method is expressed mathematically as

$$\arg \min_{(\hat{\beta_0}, \hat{\beta_1})} \sum_{i=1}^{n} \left(y_i - (\hat{\beta_0} + \hat{\beta_1} x_i)\right)^2 \tag{2.4.4}$$

To find these estimates we need to solve the system

$$\frac{\partial}{\partial \hat{\beta_0}} \sum_{i=1}^{n} \left(y_i - (\hat{\beta_0} + \hat{\beta_1} x_i)\right)^2 = 0 \tag{2.4.5}$$

and

$$\frac{\partial}{\partial \hat{\beta_1}} \sum_{i=1}^{n} \left(y_i - (\hat{\beta_0} + \hat{\beta_1} x_i)\right)^2 = 0 \tag{2.4.6}$$

Hence (James et al., 2013)

$$\hat{\beta_1} = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{2.4.7}$$

and

$$\hat{\beta_0} = \overline{y} - \hat{\beta_1} \overline{x} \tag{2.4.8}$$

where $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means

Thus the optimal least squares estimates for the regression coefficients are

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2.4.9}$$

and

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1\overline{x} \tag{2.4.10}$$

However, the least square technique sometimes performs poorly in both prediction and interpretation (Zou and Hastie, 2003). It may not always give the best estimates since sometimes solving the systems of Equations (2.4.5) and (2.4.6) might be computationally hard and when we have very large data there might be memory constraints. We will therefore discuss another optimization technique that is gradient descent for neural networks in Section 3.3 of Chapter 3

2.5 Regularization

In the least-squares estimation method, the estimates for the regression coefficients, in particular, β_0 and β_1 can tend to be biased and have high variance. There is therefore the need to balance this bias and variance. Regularization is a form of regression that can be used to solve this problem where an extra term is added to the RSS. This prevents overfitting by reducing the regression coefficient estimates (James et al., 2013) and hence reduce the variance. Overfitting will be thrown more light in Section 3.5. These methods improve the least-squares estimation method (Zou and Hastie, 2003). There are three types of regularization. Namely the L1 or Least Absolute Shrinkage and Selection Operator (LASSO), L2 or Ridge regression and Elastic net regression.

2.5.1 Ridge Regression.

Ridge regression is a form of regularization where the estimates for the regression coefficients, β_0 and β_1, are computed by adding the regularization term $\lambda_1 \sum_{j=1}^{p} \beta_j^2$ to the RSS. That is minimizing over the β_i's Equation (2.5.1) below (Hoerl and Kennard, 1970)

$$\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_i x_i)\right)^2 + \lambda_1 \sum_{j=1}^{p} \beta_j^2 \tag{2.5.1}$$

where $\lambda_1 > 0$ determines the regularization strength, p is the number of features. Ridge shrinks the β_i's for $i = 1, \ldots, p$ continuously towards 0 but never becomes 0 (Zou and Hastie, 2003), that is it penalises steeper slopes and makes the best fit line flatter.

Hence for a simple linear regression since we have just 1 feature, Equation (2.5.1) can be simplied to

$$\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i)\right)^2 + \lambda_1 \beta_1^2 \tag{2.5.2}$$

.

2.5.2 LASSO Regression.

LASSO regression was introduced by Tibshirani (1996) (Tibshirani, 1996), it penalises smaller weights and shrinks them to 0, (Zou and Hastie, 2003) hence it removes less importance features in our model. It is very useful in feature selection (Zou and Hastie, 2003) and it works best when the model contains non-important features. With LASSO the estimates for the regression coefficients are computed by adding the regularization term $\lambda_2 \sum_{j=1}^{p} |\beta_i|$ to the RSS, where $\lambda_2 > 0$ is the regularization strength. That is minimizing Equation (2.5.3) below over the β_i's (Tibshirani, 1996)

$$\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_i x_i)\right)^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j| \tag{2.5.3}$$

For a simple linear regression, Equation (2.5.3) can be simplied to

$$\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_i)\right)^2 + \lambda_2 |\beta_1| \tag{2.5.4}$$

2.5.3 Elastic Net Regression.

The elastic net regression is a type of regularization which combines both the LASSO penalty and the ridge penalty, with the aim of combining both the strengths of LASSO and Ridge to give better results (Zou and Hastie, 2003). The estimates for the regression coefficients are calculated by minimizing over the β_i's Equation (2.5.5) below (Zou and Hastie, 2003)

$$\sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_i x_i)\right)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \tag{2.5.5}$$

Ridge and LASSO regression are therefore special cases of elastic net where $\lambda_1 = 0, \lambda_2 > 0$ and $\lambda_1 > 0, \lambda_2 = 0$ respectively.

2.6 Regression Model Evaluations

The performance of a regression model can be evaluated in several ways by making use of the residual values $(y_i - \hat{y}_i)$. Some of these performance measures are

2.6.1 Mean Square Error (MSE).

MSE is the average of the squares of the residuals. This is similar to the variance of the residual values with a slight bias. MSE is always positive and is represented mathematically by Equation (2.6.1) below.

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.6.1}$$

The closer the predicted values are to the actual values the smaller the value of the MSE and vice versa.

2.6.2 Root Mean Square Error (RMSE).

As the name implies RMSE is the square root of the MSE. This is similar to the standard deviation of the residuals with a slight bias. That is using Equation (2.6.2) below.

$$\sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{2.6.2}$$

2.6.3 Mean Absolute Error (MAE).

MAE is the arithmetic average of the magnitude of the residual values. It is given by Equation (2.6.3).

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.6.3}$$

2.6.4 Coefficient of Determination (R^2).

R^2 measures the proportion of changes in the label that is explained by the features. This indicates how strong the linear relationship is between the features and labels. It is calculated using Equation (2.6.4) below.

$$1 - \frac{\text{RSS}}{\text{TSS}} \tag{2.6.4}$$

where RSS $= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ and TSS $= \sum_{i=1}^{n}(y_i - \overline{y})^2$ and $0 \leq R^2 \leq 1$.

An R^2 of 1 indicates the model is a best fit for the data, in other words the regression explains a greater change in the explanatory variable and vice versa for an R^2 of 0.

2.6.5 Adjusted R^2.

The adjsuted R^2 is where the R^2 is adjusted for degrees of freedom in the model. This is calculated using Equation (2.6.5) below.

$$1 - \frac{\text{var(r)}}{\text{var(y)}} \tag{2.6.5}$$

where var(r) $= \frac{\text{RSS}}{n-p-1}$, var(y) $= \frac{\text{TSS}}{n-1}$.

3. Artificial Neural Network

Artificial Neural Network (ANN) is a key development in the Machine learning field which is inspired by the human brain (Nwankpa et al., 2018). That is the neurons and how they are linked to each other and designed to recognise numerical patterns. ANNs are algorithms for learning and optimisation built on modelling neurons mathematically. These neurons are made up of three components known as the synapses or the connecting links which carry the weights w_i for the input values x_i for $i = 1, ..., n$, the adder which finds the sum of the weighted input values and a bias and an activation function which gives the value of the adder (weighted sum) its output value (Pattanayak et al., 2017) as seen in Figure 3.1 below.

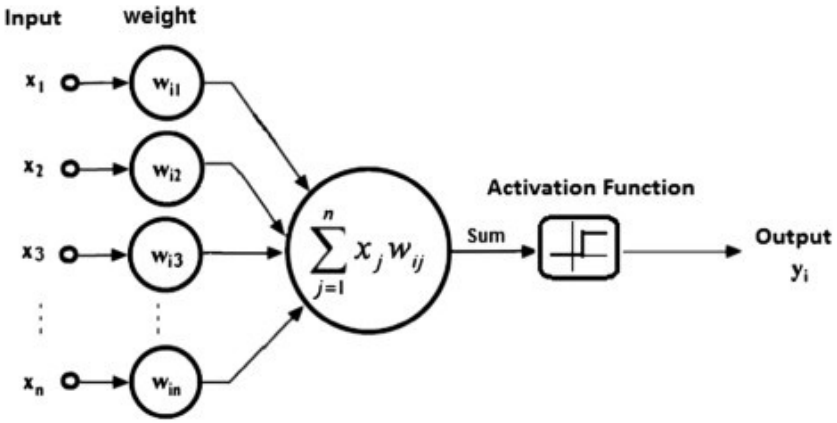

Figure 3.1: Structure of an artificial neuron with n inputs (Shubh, 2017)

ANNs are structured with an input layer that receives input values in the form of data, hidden layers that process the input values and an output layer that returns the output values. Figure 3.2 below shows the architecture of a neural network.

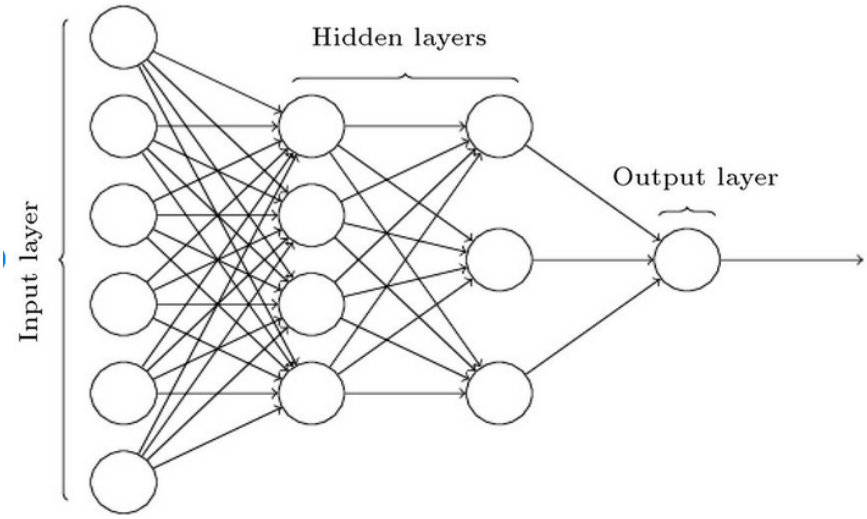

Figure 3.2: Architecture of a neural network (Salimi et al., 2017)

Artificial Neural Networks are used in several machine learning approaches like the Feed-forward Neural Network (FFNN), the Recurrent Neural Network (RNN) and the Convolutional Neural Network (CNN). However, in this project we will concentrate more on Feed-forward Neural Network.

This chapter emcompasses what Feed-forward Neural Networks are in Section 3.1 and data pre-processing and hyperparameter tuning in Section 3.2. How Feed-forward Neural Networks learns in Section 3.3, activation functions in Section 3.4 and some problems in training these networks in Section 3.5.

3.1 Feed-forward Neural Network

The Feed-forward Neural Network (FFNN) is an ANN where the various links between the nodes of each layer are directly linked with each other without forming loops or cycles in the network. FFNN was the first and most simple type of ANN devised. In this network, data is propagated in only one direction from the input nodes of the input layer through to the hidden nodes and then to the output nodes of the output layer (Sandberg et al., 2001). Single-layer Perceptron (SLP) which contains just one input and output layer and Multi-layer Perceptron (MLP) which contains an input layer, one or more hidden layers and an output layer are some classes of this network. The MLP allows for modelling complex decisions and datasets and used for regression problems. The regression performance measures especially the MSE discussed in Section 2.6 of Chapter 2 are also used in the MLP as loss functions. We will used the MLP in this essay.

3.2 Data Pre-processing and Hyperparameter Tuning

3.2.1 Data Pre-processing.

Data preprocessing comprises of the preparation and cleaning of the data to be trained on the model. In training a neural network model, the datasets are split into training data and testing data and in some cases a validation set.

Data scaling: The values of the datasets can be scaled to be between a certain range before the model is being trained. Data scaling can be achieved by standardisation the values to have a mean of 0 and a standard deviation of 1 or normalising the data values to be between 0 and 1. Standardisation and normalisation improves on the speed of the model training and reduces the probability of getting caught in a local minimum cost function.

Thus for an input x_i, it can be standardised as follows:

$$\frac{x_i - \mu}{\sigma} \tag{3.2.1}$$

where μ is the mean and σ is the standard deviation given by

$$\mu = \frac{1}{n}\sum_i x_i \quad \text{and } \sigma = \sqrt{\frac{\sum_i (x_i - \mu)^2}{n-1}} \tag{3.2.2}$$

where n is the number of training observations.

Then, normalisation is calculated using

$$\frac{x_i - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{3.2.3}$$

where x_{min} is the minimum of the values and x_{max} is the maximum of the values.

3.2.2 Hyperparameter Tuning.

The parameters of the model refer to the weights and the biases. However, the hyperparameters are the parameters that determine the final value of the weights and biases. Unlike the parameters, the hyperparameters can be specified. Examples are the number of iterations (epochs), the number of hidden layers and units, the choice of the activation function and the batch size. The hyperparameters are tuned to obtain the optimal parameters in order to improve the performance of the model. The tuning can be done in four ways. These are manual search, grid search, random search or bayesian optimisation approach (Brownlee, 2016).

Manual search: involves guessing the parameters and tweaking the hyperparameters manually based on some knowledge of the problem. This process is repeated until the parameters that give better results are obtained.

Grid search: in place of tuning the parameters manually, it can be done by cross-validation. In cross-validation, a range of values for the parameters to be tuned are specified. The model is then trained and evaluated on each combination of the specified parameters to obtain which parameters are optimal. This method can take a very long-running time depending on the data sets and the various parameters. GridSearchCV in the sklearn Python library can be used for this.

Random search: Random search is similar to the grid search. The difference is that in random search the combinations of the parameters are selected at random until the optimal parameters for training the model are obtained. Also, random search can be done using the RandomizedSearchCV model in the sklearn library.

Bayesian optimisation: Bayesian optimisation focuses on improving the other tuning types. It makes use of information derived from any experiment to know how to tune or adjust the hyperparameters for the next experiment.

3.3 How Feed-forward Neutral Networks Learn

The learning process for NN is the process of estimating the model parameters in order for the model to perform specific tasks (Nwankpa et al., 2018). Back propagation proposed by Rumelhart (Rumelhart et al., 1986) is the algorithm used in training MLP. This algorithm involves repeating two processes. These processes are discussed in the subsequent subsections.

3.3.1 Forward Propagation.

Forward propagation involves computing the values of all outputs of all the neurons in the network. Beginning with the first hidden layer each neuron output is calculated and the relevant activation functions are applied. The outputs are then used as inputs for the next hidden layer. The forward process continues from the input layer until we get to the output layer, this is referred to as forward propagation (Kubat, 2017). This phase ends with the computation of the cost function or the RSS. Thus the output of the current layer to that of the previous layer can be related by

$$a_j^l = g(z_j^l) \tag{3.3.1}$$

where a_j^l and z_j^l are the activation value (output) and the weighted sum of the inputs of j^{th} neuron in the l^{th} layer respectively and $g(.)$ is the activation function

Hence Equation (3.3.1) can be written as

$$a_j^l = g(\sum(w_{jk}^l.a_k^{l-1}) + b_j^l) \tag{3.3.2}$$

where w_{jk}^l is the weight of the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer, b_j^l is the bias of the j^{th} neuron of the l^{th} layer.

For instance given the weights w_{jk}, and input values x_i for $i = 1, 2, ..., n$ then the input to the first neuron of the first hidden layer is given as

$$g(\sum_{i=1}^{n} x_i.w_{j1} + b_j). \tag{3.3.3}$$

3.3.2 Backward Pass/Back Propagation.

Back propagation refers to the algorithm for computing the gradient of the cost or error function of the FFNN with respect to the weights and biases of the network. It involves calculating the errors for each of the output and hidden layers and then updating the weights and biases using optimisation techniques to minimize the errors until a global minimum is reached for each output node. The cost function is expressed as

$$C = \sum_{j=1}^{n}(a_j^L - y_j)^2 \tag{3.3.4}$$

where y_j is the actual output and a_j^L is the activation value of the output layer.

This gradient is computed by finding the derivatives $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$. By applying the chain rule and computing the derivatives for the output layer we obtain the gradients:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial C}{\partial z_j^L}.\frac{\partial z_j^L}{\partial w_{jk}^L} \tag{3.3.5}$$

Now, we denote $\frac{\partial C}{\partial z_j^L}$ by δ_j^L and then substituting $\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$ into Equation (3.3.5) above we have

$$\frac{\partial C}{\partial w_{jk}^L} = \delta_j^L.\, a_k^{L-1} \tag{3.3.6}$$

Now, applying the same chain rule for the derivative $\frac{\partial C}{\partial b_j^L}$ we have

$$\frac{\partial C}{\partial b_j^L} = \frac{\partial C}{\partial z_j^L}.\frac{\partial z_j^L}{\partial b_j^L} \tag{3.3.7}$$

But $\frac{\partial z_j^L}{\partial b_j^L} = 1$ hence Equation (3.3.7) gives

$$\frac{\partial C}{\partial b_j^L} = \delta_j^L \tag{3.3.8}$$

Equations (3.3.6) and (3.3.7) are therefore used in updating the weights and biases respectively for the output layer.

Futhermore the gradients for the hidden layers are computed in a similar manner by using the chain rule. For the derivative of the cost function with respect to weights for the hidden layers we have

$$\frac{\partial C}{\partial w_{ki}^{l-1}} = \frac{\partial C}{\partial z_k^{l-1}} . \frac{\partial z_k^{l-1}}{\partial w_{ki}^{l-1}} = \frac{\partial C}{\partial a_k^{l-1}} . \frac{\partial a_k^{l-1}}{\partial z_k^{l-1}} . \frac{\partial z_k^{l-1}}{\partial w_{ki}^{l-1}} \tag{3.3.9}$$

But

$$\frac{\partial a_k^{l-1}}{\partial z_k^{l-1}} = g'(z_k^{l-1}), \quad \frac{\partial C}{\partial a_k^{l-1}} = \sum_j \frac{\partial C}{\partial z_j^l} . \frac{\partial z_j^l}{\partial a_k^{l-1}} = \sum_j \delta_j^l w_{jk}^l \text{ and } \frac{\partial z_k^{l-1}}{\partial w_{ki}^{l-1}} = a_i^{l-2} \tag{3.3.10}$$

Now substituting Equation (3.3.10) into Equation (3.3.9) gives

$$\frac{\partial C}{\partial w_{ki}^{l-1}} = a_i^{l-2} g'(z_k^{l-1}) \sum_j \delta_j^l w_{jk}^l \tag{3.3.11}$$

Now, let the local gradient $\frac{\partial C}{\partial a_k^{l-1}} . \frac{\partial a_k^{l-1}}{\partial z_k^{l-1}} = g'(z_k^{l-1}) \sum_j \delta_j^l w_{jk}^l$ be denoted by δ_k^{l-1} Thus Equation (3.3.9) becomes

$$\frac{\partial C}{\partial w_{ki}^{l-1}} = a_i^{l-2} \delta_k^{l-1} \tag{3.3.12}$$

Next, computing the derivative with respect to the bias we have:

$$\frac{\partial C}{\partial b_k^{l-1}} = \frac{\partial C}{\partial a_k^{l-1}} . \frac{\partial a_k^{l-1}}{\partial z_k^{l-1}} . \frac{\partial z_k^{l-1}}{\partial b_k^{l-1}} \tag{3.3.13}$$

But $\frac{\partial z_k^{l-1}}{\partial b_k^{l-1}} = 1$, thus Equation (3.3.13) gives

$$\frac{\partial C}{\partial b_k^{l-1}} = \delta_k^{l-1} \tag{3.3.14}$$

Hence, Equation (3.3.15) seen below are the derivative used in the optimisation methods in updating the weights and biases of the models for each layer to obtain a global minimum for the loss function.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \text{ and } \frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.3.15}$$

Gradient Descent or Stochastic Gradient Descent is the optimisation method normally used in machine learning and deep learning to update the parameters for all the layers. It is used to converge towards the global minimum of the cost function in an iterative way by going along the direction of the negative gradient and updating the weight at each step (Pattanayak et al., 2017). The rate of movements is decided by the learning rate η.The learning rate determines how we converge towards the minimum. A smaller η results in a slow but steady convergence towards the global minimum whiles a bigger η results in a faster convergence but might lead to flunctuations around the minimum (Pattanayak et al., 2017). These weights and biases are updated iteratively as follows

$$w_{jk}^{(i+1)} = w_{jk}^{(i)} - \eta \frac{\partial C}{\partial w_{jk}}(w_{jk}^{(i)}) \text{ and } b_k^{(i+1)} = b_k^{(i)} - \eta \frac{\partial C}{\partial b_k}(b_k^{(i)}) \tag{3.3.16}$$

Gradient Descent (GD) updates these parameters by using all the samples in the whole training sets for a single update. This process is repeated for all training sets depending on the number of training epochs. However Stochastic Gradient Descent (SGD) does this update of the parameters by randomly splitting the training data into subsets. The updates are then done using one training subset per epoch. Thus if we have a huge number of training samples, the GD takes a longer time to converge whiles the SGD converges faster.

3.4 Activation Functions

Activation functions are functions used by ANNs to execute diversified computations between the hidden layer and output layer of an ANN architecture (Nwankpa et al., 2018), it ascertains what the output of the neuron should be given an input value. Activation functions can be grouped into three types. These are:

3.4.1 Binary Step Function.

This activation function is a threshold type of activation function. The output value is 1 when the input value is above that threshold otherwise the output value is 0 (Pattanayak et al., 2017).

3.4.2 Linear Activation Function.

This function outputs the input value directly, the output is linearly dependent on the input (Pattanayak et al., 2017). This is analogous to taking the weighted inputs with no activation function applied. The output for this function is

$$\sum w^i_{jk}.a^{i-1}_k + b^i_j \tag{3.4.1}$$

.

The linear activation function is plotted in Figure 3.3 below.

3.4.3 Non-linear Activation Function.

Non-linear activation functions are currently the most used activations functions since they allow for complex relationships between the inputs and the outputs. It works well with backpropagation and mostly used for more complex inputs like audios, videos and high dimension data sets. Some examples are sigmoid and tanh in which the output values are squashed to be between 0 and 1 , and -1 and 1 respectively (Pattanayak et al., 2017). Another example is the Rectified Linear Unit (Relu) which outputs only positive inputs and squashes negative inputs to 0. Mathematically these functions are represented as

$$g(x) = \frac{1}{1+e^{-x}}, \quad g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{and} \quad g(x) = \max(0, x) = \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases} \tag{3.4.2}$$

for sigmoid, tanh and relu respectively. The plots for the sigmoid, tanh and relu activation functions can be seen in Figures 3.4, 3.5 and 3.6 below respectively.

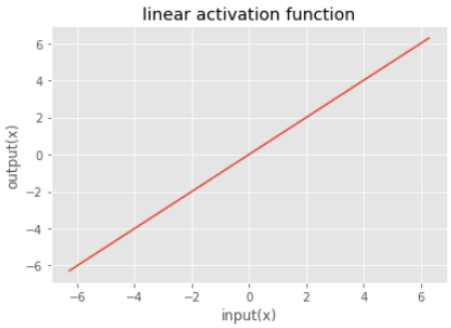

Figure 3.3: Plot of the linear activation function

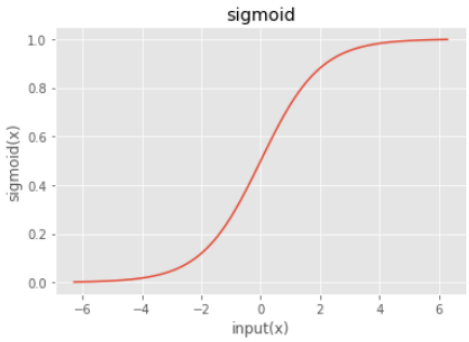

Figure 3.4: Plot of sigmoid activation function

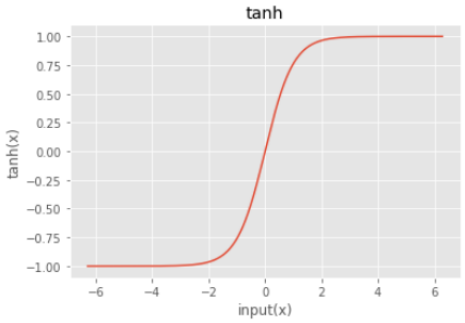

Figure 3.5: Plot of the tanh activation function

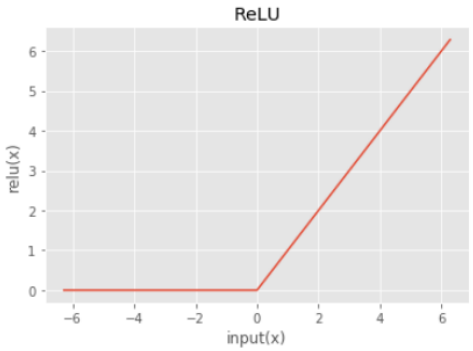

Figure 3.6: Plot of relu activation function

3.5 Problems in Training Neural Networks - Overfitting and Underfitting

The main goal of neural network models is to be able to make generalisations on new data that was not used in training the model. The models used can face several problems due to having many or too little parameters and optimisation problems and therefore causing the model to perform poorly. We will give a brief note on Overfitting and Underfitting as some causes of poor performance in the subsequent subsections.

3.5.1 Overfitting.

Overfitting is one of the biggest problems of highly parametrized models (Huang et al., 2004). This occurs when the model memorises the training data too well, learns all the noise and random fluctuations in the data. It therefore tends to do better on the training data sets than the validation data sets. Early stopping (reduce epochs) can be used to avoid overfitting (Brownlee, 2016). That is where arbitrary large epochs are specified and the model training process is halted when the loss starts increasing. The validation dataset can be used to determine when to stop if the error starts increasing. Also, the regularization techniques discussed in Section 2.5 can be used to minimise overfitting.

3.5.2 Underfitting.

Underfitting occurs when the model does not perform well on the training datasets due to too little parameters. This therefore makes the model make poor generalisations on data it has never seen. It is easy to notice by using the performance metrics. This can be curbed by trying different neural networks with different architectures and hyperparameters.

4. Experiments

We carried out several experiments in this project to compare the predictions of the CAPM with a FFNN. In this section we give details of the datasets used and the general preprocessing done in Section 4.1. The types of models we tried and their performance evaluation for the CAPM and MLP are seen in Sections 4.2 and 4.3. We then discuss the comparisons of the two models using the expected returns and the returns from the historical stock data by calculating the MSE in Section 4.4.

4.1 Datasets and Pre-processing

In this essay, we used 19-year period daily stock historical data from yahoo finance for Apple Inc., Microsoft, Amazon, Google, JPMorgan Chase and S&P500 index from January 1, 2000 to December 31, 2019. However the data for Google was from August 19, 2004. We had to set the S&P500 index to start from the same date for running models on Google stock. We utilized the adjusted closing stock price (Adj Close) column data which had 5030 observations with the exception of Google which had 3867 observations. In our adjusted closing price data we had no missing values. We used the seaborn library for visualisation to check this. The returns for Apple, Google, Amazon, Microsoft and JPMorgan were the features and that of S&P500 index was our target. The target was our proxy for the expected market returns. In calculating the returns for each of the stocks we used Equation (4.1.1) below

$$\text{Returns} = \frac{P_1}{P_0} - 1 \tag{4.1.1}$$

where P_0 is the previous adjusted closing stock price, P_1 is the current adjusted closing stock price. The expected returns are compared with the returns. The returns for all the stocks shows no autocorrelation. The matplotlib library was used to check for autocorrelation. We noticed that there were no patterns in the autocorrelation plot for all the individual stock returns. The statistics of the stock returns data used in our models are seen in Table 4.1 below

Table 4.1: Statistics of the stock returns data for the models

	Returns					
	Apple	Google	Amazon	Microsoft	JPMorgan	S&P500
Minimum	-0.5187	-0.1161	-0.2477	-0.1559	-0.2073	-0.0904
Maximum	0.1391	0.1999	0.3447	0.1957	0.2509	0.1158
Mean	0.0012	0.0010	0.0011	0.0005	0.0006	0.0002
Std	0.0254	0.0189	0.0329	0.0190	0.0242	0.0119
Skew	-1.6421	0.8952	1.1864	0.2088	0.8611	-0.0391
kurtosis	38.4517	11.5653	15.4748	10.2156	15.7716	8.8196

4.1.1 Data scaling.

There was no need to normalize or standardise our datasets since the return values had small value ranges. The returns were between -0.5 and 0.3 as seen in Table 4.1 above.

4.2 CAPM - Linear Regression Model (Experiment 1)

The linear regression model, that is the CAPM as discussed in Chapter 2 can be specifically written as

$$\mathsf{C}_r = \beta_0 + \beta_1 \mathsf{S}_r \tag{4.2.1}$$

where the C_r is the returns at day r for the company and S_r is the S&P500 returns at day r. The S&P500 returns is our market returns proxy. Since the risk-free rate is a constant and would not affect the slope of the best fit line it can be eliminated. We therefore used the market returns instead of the market risk premium. The linear regression models, regularization models and the performance measures used were from the Sci-kit learn (sklearn) library which is an open-source Python library.

4.2.1 Data spliting.

We split the datasets as follows for this model. We used 75% of our datasets for training. The remaining 25% of our datasets were used for testing our model.

4.2.2 Results and discussions.

We obtained the following performance measures for our linear regression models and regression coefficients seen in Table 4.2 below.

Table 4.2: Performance measure for the individual linear regression models and the regression coefficients

	CAPM - Linear regression model for each company stocks				
Metrics	Apple	Google	Microsoft	Amazon	JPMorgan
MSE	0.000627	0.000291	0.000199	0.000745	0.000227
R-squared	0.2094	0.3568	0.4728	0.2740	0.5782
Coefficients	Regression coefficients				
β_0	0.001005	0.000391	0.000205	0.001160	0.000347
β_1(stock beta)	1.1675	0.9572	1.0794	1.2956	1.5413

Furthermore, we implemented other regression techniques like LASSO regression, ridge regression and elastic net regression discussed in Section 2.5 of Chapter 2 to compare with the CAPM (linear regression model). These did not yield better results as expected since our model had just 1 feature. The same regression coefficients were obtained for the LASSO and elastic net regression. The LASSO gave a slope value of 0. This indicates that there is no systematic risk which is intuitively wrong. The best fit line of the linear regression models for Apple, Google, Amazon, Microsoft and JPMorgan to the testing data respectively can be seen in Figures 4.1, 4.2, 4.3, 4.4 and 4.5 below.

The R^2 values in Table 4.2 above tells us that S&P500 index return explains 57.82% changes in JPMorgan Chase stock returns with is better than all the other stocks. Followed by Microsoft with 47.28%, Google with 35.68%, Amazon with 27.40% and Apple being the least with 20.94%. The regression coefficients show that S&P500 index return is positively related to all the company stocks. The risk measures (beta) obtained are 1.16757, 0.9572, 1.0794, 1.2956 and 1.5413 for Apple, Google, Microsoft, Amazon and JPMorgan respectively. These values indicate that the volatility each for Apple, Microsoft, Amazon and JPMorgan stock is greater than that of S&P500 index whiles that of Google stock is less than that of S&P500 index.

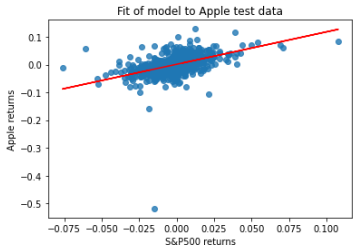

Figure 4.1: Plot of linear regression model fit to Apple test data

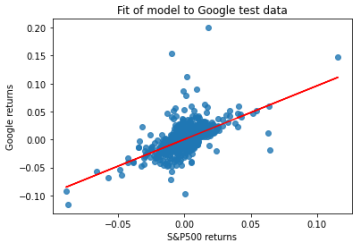

Figure 4.2: Plot of linear regression model fit to Google test data

Figure 4.3: Plot of linear regression model fit to Amazon test data

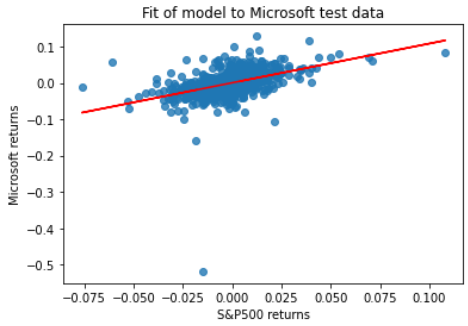

Figure 4.4: Plot of linear regression model fit to Apple test data

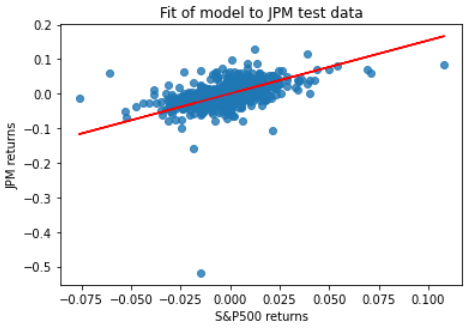

Figure 4.5: Plot of linear regression model fit to JPM test data

However, the results for the linear regression which is the CAPM does not depict a good model from the R-squared values. These results can be explained by looking at the model fit figures above. The plot of returns as seen in these figures do not necessarily follow a linear trend thus using a linear regression model would be biased and not be a good estimator.

4.3 Multi-Layer Perceptron (Experiment 2)

In this section we talk about the experiments using the Multi-layer Perceptron (MLP) and then compare the performance of the various models. We used a Feedforward Neural Network specifically the MLP since we are performing a regression analysis. In building our model, we used the Keras open-source library for neural network in Python.

4.3.1 Data splitting.

We split the returns for the neural networks as follows. We used 70% of our datasets for training our model, out of that 70% training set we used 30% for validation and the remaining 30% of our datasets were used for evaluating the model. We added a validation dataset to allow us to check for model performance more particularly overfitting of our model.

4.3.2 Hyperparameter tuning.

In MLP models, the performance of the models depends on the hyperparameters used. In the experiments we tuned our hyperparameters manually in a systematic way. We started with a simple

architecture and tune the number of epochs. We then build the current architecture from the previous by changing the hyperparameters based on the performance of the previous. We also made sure not to add too many hyperparameters in order not to overfit our model and use a simple architecture as much as possible. This is because we had a simple regression problem.

4.3.3 Results and Discussions.

In model development, its advisable to start with the simplest model and then build on that to more complex models. All the model architectures comprised of a layer with 1 neuron each for input and output since we have one feature and one target. The activation function used for the input and output layers were the linear function. Systematically, we commenced all the models with 1 hidden layer with 1 node and a linear activation function and then increased the epochs to check if there will be improvements. We then changed the hidden layer activation function to the tanh function since our values range from -0.5 to 0.3. Given this architecture we run the model on several iterations (epochs) to see if the MSE will become less and if the model performance will improve. If there are no improvements we add nodes and hidden layers simultaneously to the previous architecture. When adding the hyperparameters does not improve much on the model we take the simplest model which gave a smaller MSE and a better model (no overfitting). Since this is more efficient and it saves resources.

Considering Google stock, a 1 neuron hidden layer with linear activation increasing the number of epochs to 60 gave us a better model with a smaller MSE. Whiles running the model for more than 60 epochs our models begun to overfit. As we increased the number of nodes and hidden layers with a tanh activation function we noticed our model's performance decreasing. So the latter model was our best model for efficiency and to save space and time. The best model architecture was with 1 input, hidden and output layer with linear activation ran with 60 epochs. The plot of the training loss (MSE) per epoch is seen in Figure 4.6 below. The performance for training and validation (MSE) seen below in Figure 4.7 indicates that our model was not overfitted.

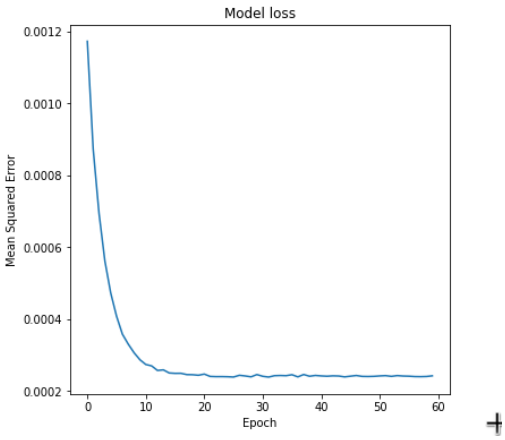

Figure 4.6: Plot of the training loss against epochs for Google

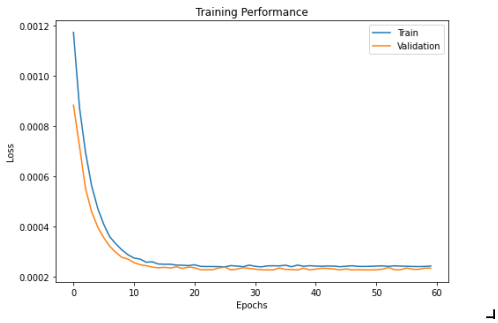

Figure 4.7: Plot of validation and training loss against epochs for Google

However, for Amazon stock with a 1 neuron hidden layer with linear function as we increased the number of epochs from 20 our results did not change much. The MSE for both the validation and testing was around the same value of 0.0012 and 0.0010 respectively. Changing the activation for the hidden layer to tanh gave some improvements which got better with increased epochs to 100. Whiles adding more nodes and layers gave a greater MSE. The best model for Amazon had a hidden layer with 1 node, a

tanh activation and 100 epochs. The plot of the loss (MSE) per epoch is seen in Figure 4.8 below. The performance for training and validation (MSE) is seen in Figure 4.9 below.

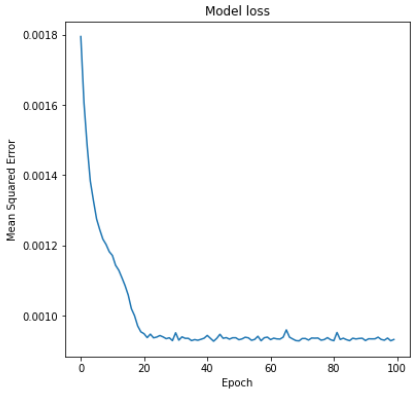

Figure 4.8: Plot of the training loss for Amazon against the epochs

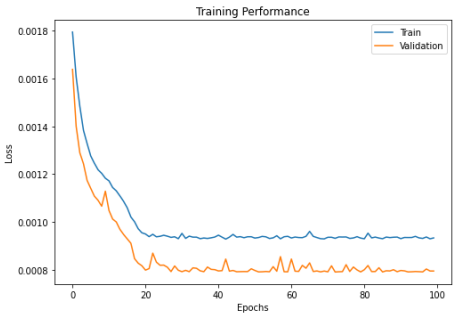

Figure 4.9: Plot of the validation and training loss for Amazon against the epochs

Furthermore, considering Apple stock, with a hidden layer (1 node and linear activation) as we increased the number of epochs from 10 to 100 epochs the MSE decreased from 0.000679 to 0.000429 on the validation datasets. Using a tanh activation function and adding more layers decreased the MSE slightly but the model was overfitting. Thus the 1 node hidden layer (linear) with 100 epochs gave the best model efficiently. Figure 4.10 below shows the training loss (MSE). The performance for training and validation (MSE) is seen in Figure 4.11 below.

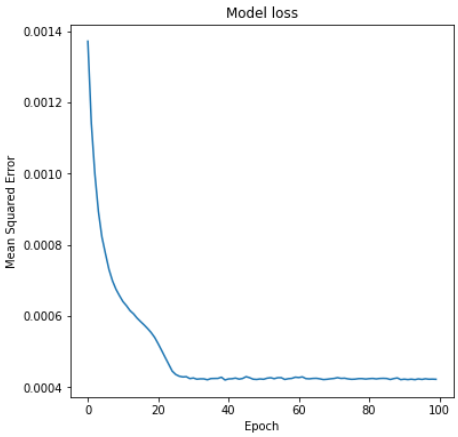

Figure 4.10: Plot of the training loss per epoch for Apple

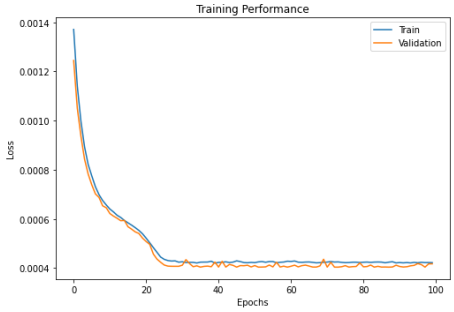

Figure 4.11: Plot of the validation and training loss for Apple against the epochs

Next, the architecture for Microsoft was similar to Amazon. With a 1 node hidden layer with linear activation increased epochs and neurons did not change our MSE much. We then used a tanh activation function which gave smaller MSE and a good model as we increased the epochs from 40 to 80. However, increasing the neurons decreased the MSE. Using a 4 neuron hidden layer with the tanh function for 80 epochs was the best we obtained. The validation loss and training loss converges closely as seen in the training and validation loss plot in Figure 4.13 below. The plot of the loss (MSE) per epoch is seen in

Figure 4.12 below.

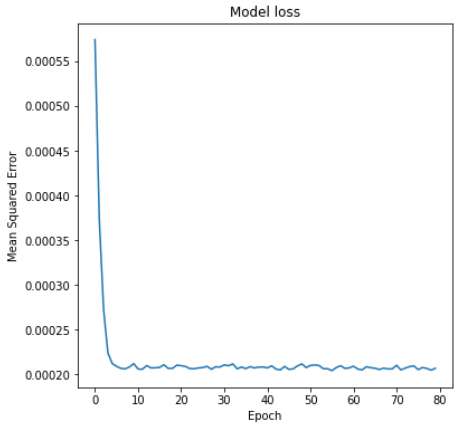

Figure 4.12: Plot of training loss per epoch for Microsoft

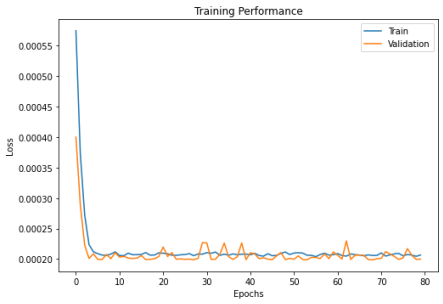

Figure 4.13: Plot of training and validation loss per epochs for Microsoft

However, for JPM with 1 linear hidden layer with 100 epochs our validation and training loss were far apart from each other. Increasing the epochs did not change our results much. Also when we increased the neurons it did not give better results. Again, adding an extra layer was giving us overfitted models even though our MSE slightly decreased. The best choice was a hidden layer with 1 neuron and a tanh activation function with 60 epochs. The validation loss and training loss converges closely as seen in the training and validation loss plot in Figure 4.15 below. The plot of the loss (MSE) per epoch is seen in Figure 4.14 below.

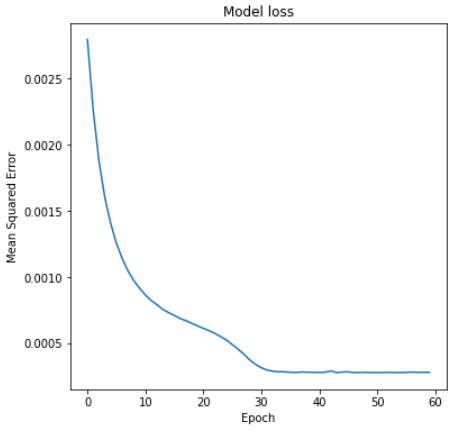

Figure 4.14: Plot of training loss per epoch for JPMorgan Chase

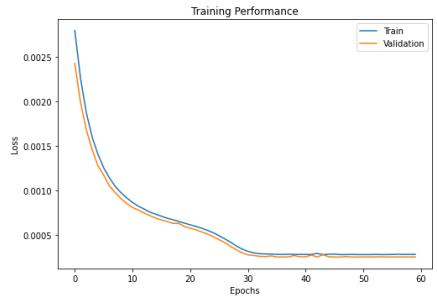

Figure 4.15: Plot of training and validation loss per epochs for JPMorgan Chase

Hence we observed that for our models more complex models with increased parameters did not give us better results. The best models were the simple models with few parameters. Also for most of our models increasing the epochs to 100 gave better models. Whiles in others 100 epochs and beyond either did not yield any change or gave greater MSE and overfitted models. The testing MSE for our best MLP models for all five stocks can be seen in Table 4.3 in section 4.4.

4.4 General Discussions

Now, comparing the CAPM and the MLP models for the individual stocks considering the MSE seen in Table 4.3 below. The MLP's MSE for all the stocks was less than the MSE for the CAPM with the exception of JPMorgan. However the difference was not much for all of the stocks. The model for JPMorgan was a good model since both the validation loss and training loss converge closely eventhough the MSE was greater. These results were as expected for the stocks with the exception of JPM. Since in the CAPM the returns are modelled with a linear relationship but the MLP allows us incorporate non-linearities and data complexities between the feature and target in the model. However for JPMorgan Chase it indicated that using the linear regression model (CAPM) was better since the MSE was less.

Table 4.3: MSE for comparison of the CAPM and the MLP for each stock

	Performance measure metrics (MSE)				
Model	Apple	Google	Microsoft	Amazon	JPMorgan
CAPM	0.000627	0.000291	0.000199	0.000745	0.000227
MLP	0.000589	0.000237	0.000180	0.000728	0.000258

5. Conclusion and Future Work

In this essay, we examined the Capital Asset Pricing Model for measuring risks and predicting returns of stocks for five companies on the NYSE. These are Apple Inc. (AAPL), Microsoft Corporation (MSFT), Alphabet Inc. (GOOGL), Amazon and JPMorgan Chase (JPM) using the S&P500 index (GSPC) as the market returns proxy. We then compared this model using the same predictor with MLP models which allows for non-linearities. We compared the models using the returns of the companies' stocks and then calculating the MSE as our loss function for the performance measure. Our results showed that the CAPM gave a greater MSE compared to the MSE for the MLP for all of the stocks except for JPMorgan. This indicates that using the CAPM for JPMorgan Chase was better that relaxing the linear assumption between the returns and using the MLP. The MLP's performance was influenced by the number of epochs, the number of neurons, the activation function and the simplicity of the model. For almost all the models, 1 hidden layer with either linear or tanh activation function resulted in better models. It can therefore be concluded that for most of the stocks MLP outperforms the CAPM in predicting returns even though the CAPM is easy to interpret. This can be attributed to the fact that the MLP allows the modelling of non-linearities and complexities unlike the CAPM where only a linear relationship is expressed between the returns (feature and target).

A possible extension of this work could be to analyse and determine under what conditions the MLP outperforms the CAPM. Also further possibilities could be to either compare the MLP approach to a Recurrent Neural Network (RNN) approach in stock returns predictions using the NYSE or a different Stock Exchange for a developing economy.

Acknowledgements

I would like to thank the Almighty Allah for his mercies and undying love throughout my life and seeing me through this endeavour till the end. It would not have been possible without Him.

My sincere and utmost gratitude also goes to the AIMS team and it's funders for supporting this work and giving me the chance to be a part of this wonderful and life changing journey and family. I would also want to say thank you to all my AIMS friends for listening, assisting and making this possible.

A big thank you to my supportive supervisor, Prof Kanshukan Rajaratnam as well as my wonderful and contributory tutor, Rock Koffi for their endless aid and motivation to work hard through out this project. Words can not describe how grateful I am for their steady and consistent guidance.

I would also want to express my heartfelt gratitude to my family especially my mom Mrs Ayishetu Abdullah and my dad Mr Larabu Hijo for being my motivation and seeing me through my education till date. This project would not have been possible without their constant prayers, guidance and counselling.

References

Brownlee, 2016. Machine learning mastery. machinelearningmastery, https://machinelearningmastery.com/start-here/, Accessed May 2020.

Cao, Q., Parry, M. E., and Leggio, K. B. The three-factor model and artificial neural networks: predicting stock price movement in china. *Annals of Operations Research*, 185(1):25–44, 2011.

Diksha, n.d. CAPM:analysis and limitations. economicsdiscussion, http://www.economicsdiscussion.net/portfolio-management/capm/capm-assumptions-and-limitations-securities-financial-economics/29904, Accessed April 2020.

Erdinç, Y. Comparison of capm, threefactor fama-french model and five-factor fama-french model for the turkish stock market. *Financial Management from an Emerging Market Perspective*, pages 69–92, 2017.

Fama, E. F. and French, K. R. The capm: Theory and evidence. *Center for Research in Security Prices (CRSP) University of Chicago Working Paper*, (550), 2003.

Fama, E. F. and French, K. R. A five-factor asset pricing model. *Journal of Financial Economics*, 116 (1):1–22, 2015.

Frees, E. W. *Regression modeling with actuarial and financial applications*. Cambridge University Press, 2009.

Goetzmann, W. N., Brown, S. J., Gruber, M. J., and Elton, E. J. Modern portfolio theory and investment analysis. *John Wiley & Sons*, 237, 2014.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

Hoerl, A. E. and Kennard, R. W. Ridge regression: applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

Huang, Z., Chen, H., Hsu, C.-J., Chen, W.-H., and Wu, S. Credit rating analysis with support vector machines and neural networks: a market comparative study. *Decision support systems*, 37(4):543–558, 2004.

James, G., Witten, D., Hastie, T., and Tibshirani, R. *An introduction to statistical learning*, volume 112. Springer, 2013.

Kisman, Z. and Restiyanita, S. M. the validity of capital asset pricing model (capm) and arbitrage pricing theory (apt) in predicting the return of stocks in indonesia stock exchange. *American Journal of Economics, Finance and Management*, 1(3):184–189, 2015.

Kubat, M. *An introduction to machine learning*, volume 2. Springer, 2017.

Lintner, J. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. In *Stochastic optimization models in finance*, pages 131–155. Elsevier, 1975.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Pattanayak, S., Pattanayak, and John, S. *Pro Deep Learning with TensorFlow*. Springer, 2017.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Salimi, A., Erdem, O. A., and Rafighi, M. Applying a multi sensor system to predict and simulate the tool wear using of artificial neural networks. *Scientia Iranica*, 24, 08 2017. doi: 10.24200/sci.2017.4247.

Sandberg, I. W., Lo, J. T., Fancourt, C. L., Principe, J. C., Katagiri, S., and Haykin, S. *Nonlinear dynamical systems: feedforward neural network perspectives*, volume 21. John Wiley & Sons, 2001.

Sattar, M. Capm vs fama-french three-factor model: an evaluation of effectiveness in explaining excess return in dhaka stock exchange. *International Journal of Business and Management*, 12(5):119, 2017.

Sharpe, W. F. Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19(3):425–442, 1964.

Shubh, 2017. Artificial neural networks (basics). becominghuman, https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c, Accessed May 2020.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Womack, K. L. and Zhang, Y. Understanding risk and return, the capm and the fama-french three factor-factor model. *Tuck Case*, pages 3–111, 2015.

Zou, H. and Hastie, T. Regression shrinkage and selection via the elastic net, with applications to microarrays. *JR Stat Soc Ser B*, 67:301–20, 2003.