UNIVERSITY OF BIRMINGHAM

Using Artificial Neural Networks to Model the Non-Linear Relationships between the Fundamental Factors and the Performance of a Portfolio

by

Dhanish Ashraf
1336289

A thesis submitted in partial fulfillment for the
degree of Master of Computer Science

in the
College of Engineering and Physical Sciences
School of Computer Science

Supervised by Dr. Shan He

September 2017

UNIVERSITY OF BIRMINGHAM

Abstract

College of Engineering and Physical Sciences
School of Computer Science

by Dhanish Ashraf

This paper aimed to investigate the hypothesis that artificial neural networks can be used to model the non-linear relationships between the fundamental factors and the performance of a financial portfolio. A feed-forward network with back propagation was used. The data used was taken from 25 years of financial statements of the current Financial Times stock exchange top 350 companies and was pre-processed in order for it to be used for machine learning. A random search was used for hyper-parameter optimization. The model was trained quarter by quarter. The results obtained by the artificial neural networks proved, using statistical methods, the results to be significantly more accurate than other machine learning methods and hence, the hypothesis to be true.

Acknowledgements

I would first like to thank Dr. Shan He, for his advice and supervision during the length of this project. I am also particularly grateful to Dr. Allan White who kindly agreed to help with the financial aspect of this project and provide the financial data used to train the models.

Contents

List of Figures

Chapter 1

Introduction

The aim of the work described in this report was to investigate the hypothesis that artificial neural networks can be used to model the non-linear relationships between the fundamental factors and the performance of a financial portfolio. The fundamental factors refer to the quantitative and qualitative information about a company that help traders calculate the intrinsic value of a stock.

As of yet, published results have not been found to confirm or deny the effective use of neural networks and fundamental factors to model the performance of a portfolio. Previous studies using artificial neural networks to predict the stock market have a tendency to overlook the fundamentals. Alternatively, these studies focus on other factors such as the intraday stock prices, volumes of stocks and even the news. Despite the lack of research directly related to our hypothesis, we do encounter papers that confirm that fundamental analysis can indeed be used to find the true value of securities. The aforementioned papers, which will be studied more closely in the literature review, raise the question as to why powerful machine learning models, such as artificial neural networks, and fundamental analysis have not been used to model the performance of financial portfolios.

In order to find the complex relationships between the fundamental factors and the performance of a portfolio, an adequate amount of data was required. In reflection of this, we used 25 years worth of quarterly financial statements for the companies in the Financial Times Stock Exchange 350 (FTSE 350). In particular, we look at the quantitative data outlined in companies quarterly financial statements and using a feed-forward

network with back propagation, we investigate whether relationships exist between the historical data and their quarterly performances. The relationships found are used to predict the performances of the stocks from which we compiled five virtual portfolios, with the first portfolio being the best predicted performing stocks and the last portfolio being the worst predicted performing stocks. We compare the predicted performances of the portfolios to the actual performances in order to test the accuracy of the predictions. Accurate predictions would indicate that the neural network has found non-linear relationships between the fundamentals and the performance of stocks. However, non-accurate predictions do not necessarily indicate that the network did not find any relationships. By performing a Wilcoxon signed-rank test using the actual performances of the best and worst portfolios, it can be found if there is a statistical significance between the portfolios. The test confirms whether the difference in performances between portfolios happened by chance or such differences will likely occur again. If there is a statistical difference, then this can tell us that whilst the predictions may not be very accurate, the network has still been able to learn details of the relationships and by further developing the network, the predictions may become more accurate. In order to validate the choice of a neural network over other common machine learning algorithms, we compare our results to that of support vector machines and linear regressors to see whether they outperform the neural network .

This report consists of 7 main chapters. In order to understand this work, a working understanding of the stock market, fundamental analysis and machine learning concepts is required.

In particular, the reader should have an understanding of the stock market, fundamental analysis, basic machine learning algorithms as well as concepts and most importantly, how artificial neural networks work. These concepts are covered in detail in Chapter 2. Once the reader is familiar with the background concepts, Chapter 3 analyses previous studies using artificial neural networks to predict the stock market as well as papers in support of fundamental analysis. Following the support for our hypothesis outlined in previous publications, Chapter 4 describes the problem specification and the rationale behind the requirements. It was necessary to undertake vast data pre-processing before the data was usable, which is described fully in Chapter 5. Following this, Chapter 6 looks at how the neural network itself was implemented and the rationale for the design choices. Finally, in Chapter 7 we go onto analyze the results of the neural network and

compare these results against other machine learning algorithms. Areas for improvement and further development within this field are also discussed.

Chapter 2

Background

This section provides the necessary background knowledge required to understand this project.

2.1 The Stock Market

2.1.1 What is it?

The outline presented here is based on Investopedias article on the stock market [1].

The stock market, or equity market, consists of a group of markets and exchanges where various securities such as stocks and bonds are bought and sold, or more formally, issued and traded. Securities are traded either on a formal exchange such as the London Stock Exchange or over-the-counter (i.e. through a dealer network). The stock market allows investors to gain partial ownership of a company in exchange for giving companies capital. In other words, the stock market is a place for companies to raise capital and investors to buy a percentage of a company.

When a private company needs to expand or simply needs more money, they may choose to start offering stocks to the public, known as Initial Public Offerings. If a stock is being sold on a market to the public for the first time, they are sold on the primary market. Once these stocks have been sold, any future trading is done on the secondary market. The primary market is where the company in question will raise its capital. The company will not receive any money from the trading of its shares on the secondary market.

The stock market has many indexes which indicate how the market or a section of the market is performing overall. Indexes include an index for the market as a whole or for specific portions of the market such as The Dow Jones Industrial Average which is the price weighted average of the 30 largest companies in the United States. Indexes act as very practical benchmarks against which traders may compare their own group of financial assets, otherwise known as their portfolio.

The stock market is important because it enables companies to raise money by offering stocks and corporate bonds (a debt security where the investor lends money to a company at some interest rate for a specified time before claiming their money back). In return investors receive a percentage of the company's profit, known as a dividend. If the value of the company increases, investors may profit by trading the shares at this appreciated value.

2.1.2 Financial Times Stock Exchange Group

The London Stock Exchange (LSE) is the main stock exchange in the United Kingdom, the largest in Europe and the third largest in the world by market capitalization. It contains 350 companies from over 50 different countries [2].

The Financial Times Stock Exchange (FTSE) Group is an independent organization that is owned by both the Financial Times and the LSE. The FTSE Group create and manage indices of shares which tracks the markets changes over time. The most well known index is the FTSE 100 which is made up of the 100 largest companies by market capitalization, which is around 80% of the entire market capitalization, listed on the LSE [3]. Market indices measure the value of groups of stocks and help investors track market changes over long periods of time. For example, if the FTSE 100 index continues to decrease over a period of time then an investor might conclude that companies within the index are struggling and decide not to invest in them [4]. The companies within the FTSE 100 are assessed each quarter and if their shares have under-performed then these companies may drop out of the FTSE 100 and be replaced by another company whose shares have performed better.
Another well known index is the FTSE 250, which is made up the of next 250 largest companies in the LSE after the FTSE 100.
The FTSE 350 is an index made up of the FTSE 100 and FTSE 250 combined. In other

words, it is an index containing the 350 largest companies by market capitalization listed on the LSE.

2.1.3 Fundamental Analysis

The outline presented here is based on Investopedias guide to fundamental analysis [5].

Fundamental analysis is a technique that attempts to determine a security's true value, its intrinsic value, by focusing on underlying factors that affect a company's actual business and its future prospects, called the 'fundamentals'. It can be performed on a company, an industry or the economy as a whole. By looking at the fundamentals one tries to answer questions such as whether the company's revenue is growing, if they can beat their competitors and if they can repay their debts.

Fundamental Analysis has two main assumptions:

1. The price of the stock on the market does not reflect the intrinsic value of the stock.
2. The price of the stock on the market will eventually reflect its true intrinsic value. This could take days or years.

Fundamentals are the 'things' that are being researched to calculate the intrinsic value of a security. They can be split into two categories; quantitative fundamentals and qualitative fundamentals.

Quantitative fundamentals are numeric, measurable quantities taken from financial statements. There are three main financial statements; balance sheets, income statements and cash flow statements.
The balance sheet outlines the company's assets, liabilities and equity at a specific point in time. The assets are the resources - such as the buildings, machinery and cash - that the company own or have control of. The company's liabilities are its debts, which they have to pay back. The equity is the total amount of money that the owners have contributed to the company. The balance sheet tells investors a lot about the company's fundamentals, for example, it will tell them how much debt the company are in, what kind of funds the company has generated and, how much cash and equivalents it has.

The income statement measures a company's performance over a specific period of time; usually quarterly or annually. It will give information such as the revenues, earnings and, earning per share for that period of time. The income statement lets investors know how well the company is performing and whether they are making any profit. If, for example, a company has low expenses compared to its revenue then this usually points to strong fundamentals.
The cash flow statements outline the company's cash related activities. Examples include the operating cash flow which is the cash generated from day to day business, the cash used for investing in assets, and the cash paid or received from the issuing and borrowing of money. Cash flow statements are important in assessing the fundamentals because it shows how the company funds its operations and how the company can pay for their future growth.

Qualitative fundamentals are factors relating to the business that can not be measured. For example, company management is an important qualitative fundamental since they are the people that make the key decisions that will really impact the company and its success. Therefore studying the management, their history and even booking meetings with them can give the investor a better idea of whether they have the ability to steer the company to financial success. Another example includes researching a company's customers; i.e. whether they have millions of customers or if all their business come from one main customer. If the company had one main customer, the government, say, an investor would want to know if a change in law, resulting in the loss of their only customer wipe out the entire company?

2.1.4 Stock-Picking Strategies

The outline presented here is based on Investopedias "Guide to Stock-Picking Strategies" [6].

There are many different strategies to picking the best stocks to invest in, all of which have the same aim of receiving returns greater than the market average. It should be emphasized that there is no "best strategy" since all the strategies have their own merits. Here we discuss a few popular strategies.

2.1.4.1 Value Investing

Value investing is one of the best known trading strategies and is based on fundamental analysis. The idea is simple; find companies that are trading at price lower than what they are actually worth. That is, find companies that are under-priced in the stock market according to its fundamentals and invest in them.

2.1.4.2 Growth Investing

Growth investing is also based on fundamental analysis and is a strategy where investors focus on the potential of a company. They find stocks that are trading at a price higher than their intrinsic value. They do this with the belief that the intrinsic worth of these companies will grow and so will the stock price.

2.1.4.3 Technical Analysis

Technical Analysis is the complete opposite of fundamental analysis. It is a short term strategy that selects stocks by analyzing charts which show the past prices and various indicators and then make conclusions about the stock price movements. Technical analysts make three major assumptions:

1. All information about a stock is already factored into its price,
2. The price of a stock follows a trend,
3. Historical stock price movements tend to repeat themselves.

Put simply, technical analysts select stocks by analyzing statistics generated by past market activity, prices and volumes.

2.2 Machine Learning Concepts

Here we introduce a few key concepts of machine learning that will help to understand this project. The outline presented here is based on Sebastian Raschka's blog post "Predictive modeling, supervised machine learning, and pattern classification" [7].

2.2.1 Datasets

A typical dataset contains some number of features and a label. Imagine we have a dataset containing characteristics of football players; their height, weight, dominant foot and sprint speed, each of these characteristics is a feature of the dataset. Features can be discrete (categorical) or continuous but discrete features usually need to be converted into a format that can be understood by the model. For example, when considering the dominant foot of the football player, right, both or left footed could be represented by 1, 0 and -1 respectively.
The label is the characteristic or attribute of the dataset that we try to predict which can also be continuous or discrete. In the footballer example, the label could be the number of goals a footballer will score in the season, or the team the footballer plays for. Again, the team name would have to be in a numerical format that the model can better understand, possibly by assigning a number to each possible team.
Each entry in the dataset is called a "sample". That is, given our football player dataset, each sample corresponds to all the data that we have on a single footballer. A sample can also be referred to as an "instance" or "observation".

When we have our dataset, it should randomly be split into two sets; the training set and the test set. The training set is the dataset used to train the model. The model uses this dataset to learn the relationships in the data in order to correctly predict the labels. The test set is then used to evaluate how well the model can predict the labels and should only be used once, right at the end in order to avoid overfitting. Overfitting occurs when a model predicts the labels of the test data set very well, yet performs poorly, with high prediction error, when attempting to predict unseen data.

2.2.2 Missing Data Values

If a dataset has missing values, then the machine learning algorithms will likely not perform correctly. There are various options for dealing with missing values.
If the number of missing values is low, it could be feasible to just get rid of these samples. For example, in a dataset of 1000 samples, of which 10 samples have missing values, the affect of removing these 10 samples, would likely be insignificant to the models performance.

However, if the number of samples with missing values is considerable, removing these samples would result in a much smaller dataset and therefore significantly reduce the models potential performance. In such cases, there are many imputation techniques that could applied. One technique is taking the overall mean of the values in the feature column and using that value to impute the missing values in that column. This technique is recommended for dataset samples that are independent of one another like in our football player example where the data for a particular football player would not have any affect on the data for any other footballer.
Another, arguably more superior, option is to use the k-nearest neighbour mean. The idea is the same as using the overall mean except now we only take the mean of the k-nearest neighbours of the missing value. This technique is more recommended for datasets where the values could have some dependence on one another. Consider, for example, the daily highest temperatures of a city, for data missing inside a heatwave period, the k-nearest mean would produce a better representation of missing values during that period than an overall average.

2.2.3 Data Normalization

Most machine learning algorithms require us to either normalize the data, or use other feature scaling techniques, to be able to fairly compare different features. For example, if we have a dataset about adults, containing various information such as their age, job, annual salary, etc, then the scale of ages would be in the tens whereas their salaries would likely be in the thousands. Clearly in order to fairly compare these values, the data should be on the same scale otherwise the larger values could affect how the model learns the relationships. It is important to remember to normalize the training set and test set in the exact same way. Common normalization techniques include min-max scaling and z-score standardization.

2.2.4 Classification & Regression

There are two types of predictive models, regression and classification. Both types have the same general idea; use a machine learning algorithm to learn relationships in the training dataset and then use these relationships to make predictions on unseen data. These relationships correspond to relationships between the variables and also trends in

the data.

Classification models use the relationships to assign discrete classes to the samples, or observations, in the dataset. For example, one may wish to build a predictive model that will classify whether a university is 'Good', 'Okay', or 'Bad' based on rating scores by current students.

Regression models use the relationships to assign continuous variable to the samples in the dataset. For example, one could build a model to predict students exam scores based on their assessment performances throughout the year.

2.2.5 Supervised and Unsupervised Learning

Supervised learning refers to a task where the sample labels are known. That is, for each input into the model we already know what the desired output is and we want the model to learn the general relationships in the samples to be able to predict desired outputs. An analogy to such learning may be a student revising for an exam. The student may use past papers along with their correct answers to learn how to properly answer the questions and then use this knowledge to try and correctly answer the actual, unseen, exam.

Unsupervised learning on the other hand, refers to when the sample labels are unknown. The goal is to learn the relationships between the data without the guidance of the true labels. Unsuperivsed learning usually looks for similarity measures in the data and groups, or clusters, the samples together based on the similarities.

2.2.6 Support Vector Machines

The outline presented here is based on Bruno Stecanella's blog post on MonkeyLearn "An introduction to Support Vector Machines" [8].

A Support Vector Machine (SVM) is a commonly used supervised machine learning algorithm for both classification and regression problems. The basic idea is to project, or map, our data into a higher dimension in which it becomes separable and therefore easier to find relationships between the data. However, mapping vectors into a higher dimension and then calculating the dot product can be computationally expensive. In

order to overcome this difficulty the kernel trick is introduced, which allows us to compute the dot product of the higher dimensional vectors without mapping the vectors into the higher dimensional space. Mathematically, we can define a kernel function K as

$$K(x \cdot y) =< f(x) \cdot f(y) >,$$

where x & y are m dimensional vectors, f is a function that maps a vector from an m-dimensional space to a n-dimensional space (where $n >> m$) and $(x \cdot y)$ denotes the dot product of x and y.
As we can see, to compute the dot product, $f(x) \cdot f(y)$, we first have to calculate $f(x)$ and then $f(y)$, which can be very computationally expensive if f is mapping its inputs into a much higher dimension. However, by using a kernel function, it is only necessary to compute the dot product of x and y, hence saving a lot of computational effort.
Some examples of common kernels include the linear kernel, radial bias function kernel, and polynomial kernel. A kernel is chosen depending on the shape of data.

An advantage of SVMs are that the prediction accuracy is generally high and they have a better performance than artificial neural networks when there is a limited number of samples or a high number of features. Artificial neural networks are discussed in the next section.

2.2.7 Linear Regression Model

The outline presented here is based on Dr. Jason Brownlee's blog post "Linear Regression for Machine Learning" [9].

A linear regression model is another commonly used machine learning algorithm that originates as a statistical algorithm. More specifically, it is a linear model that assumes a linear relationship between the features and the label. That is, given a sample, the label (the output) can be predicted using some linear combination of the features (the inputs). For example, given an input with 3 features, x_1, x_2, x_3, the output y, is predicted by a linear combination of the inputs, of the form

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_0,$$

where β_1, β_2 & β_3 are the coefficients of the inputs and β_0 is the bias coefficient.
The role of the coefficients is similar to that of the weights in an artificial neural network; they represent the significance of the relationships in the input data. The bias coefficient gives our regression line (the line of best fit) some degrees of freedom so that it can be moved to best fit the data.

In order to train a linear regression model, we have to find the best coefficients for our linear equation such that it can predict the outputs to a good accuracy. There are many ways to do this, such as the ordinary least squares method and gradient descent. The ordinary least squares method involves minimizing the sum of the squared residuals. The sum of the squared residuals is calculated by taking the squared distance from each data point to the regression line and then summing all the squared errors together. Gradient descent is explained in section 2.3.3 Once the coefficients have been calculated, making predictions is as simple as plugging the feature values into the equation.
A key part of training a linear regression model is that the relationship between the features and labels must be made linear if they are not already.

2.3 Artificial Neural Networks

Dr. Robert Hecht-Nielsen, the inventor of the first neurocomputer, defined Artifical Neural Networks (ANNs) as:

> "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." [10]

ANNs are inspired by the human brain, in particular, the behaviour of biological neurons. The human brain has the ability to recognize, learn and generalize hidden patterns in the world and then make informed decisions based on the available information. This is the basic idea behind ANNs; create a computer simulation that can learn and recognize patterns, generalize them and then make decisions in a human-like way.
Neurons have 3 main basic functions:

- Receive signals or information,

- Process this information,
- Pass the processed information, its output (activation), on.

The neuron receives information from the outside world or from other neurons via the dendrites (fibres which spread out from the soma, or cell body). The input signals can either cause the neuron to generate an electrical impulse (i.e. the signal is excitatory) or stop the neuron from generating an electrical impulse (i.e. the signal is inhibitory). The soma sums the excitatory and inhibitory signals and if the sum is greater than some threshold value, the neuron generates an action potential (i.e. it fires). This action potential is then passed to other neurons via the axon, or the action potential is passed to certain parts of the body, such as muscles [11]. Figure 2.1 depicts a biological neuron.

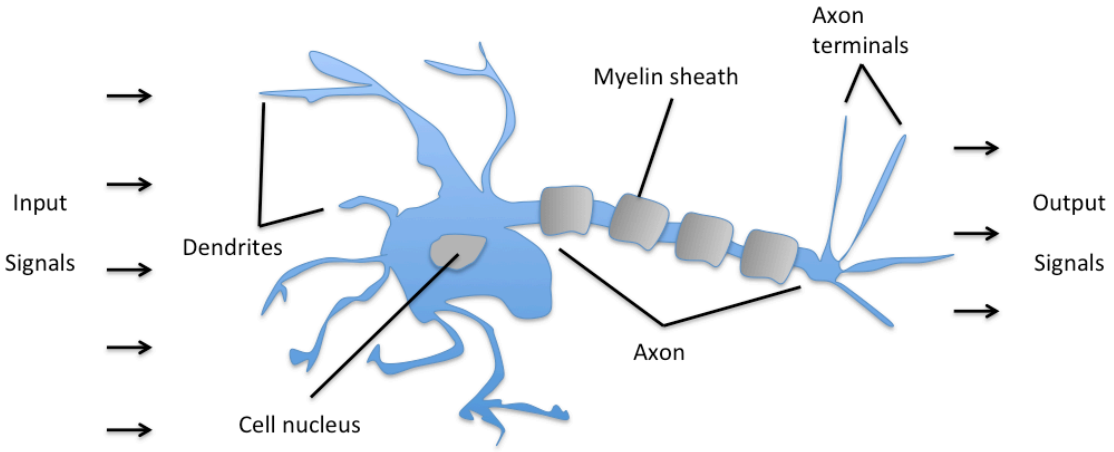

FIGURE 2.1: A Biological Neuron [12]

2.3.1 Single-layer Perceptron Model

The single-layer perceptron model is a linear classifier - a model that takes linear combinations of the inputs and makes a classification decision based on this linear combination. The perceptron model does not have many real life applications however it does form the basis for more complex multi-layered neural networks. The aim of the model is to form some linear decision boundary between classes such that given some input, it will be able to correctly classify the input based on the decision boundary. For example, in figure 2.2 the shaded circles represent one class, the empty circles represent another

class and the black line represents the decision boundary learned by the model. If the input falls on the right side of the decision boundary then it will be categorized as the shaded circles class, otherwise it is categorized as the empty circle class. The decision boundary is formed, or learned, from a training dataset. The learned decision boundary is then tested on the test dataset to see if it can be used to correctly classify unseen samples.

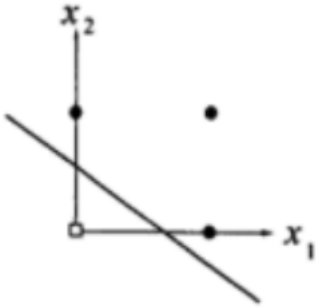

FIGURE 2.2: A linear decision boundary [13]

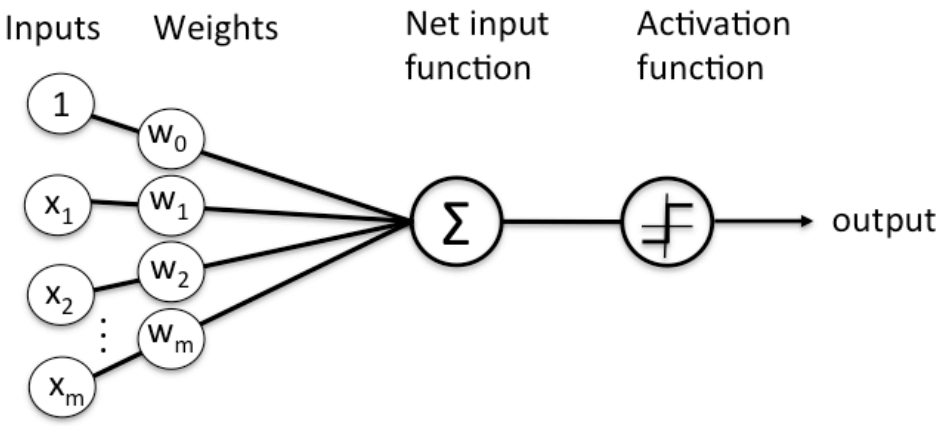

FIGURE 2.3: A single-layer perceptron [14]

The single-layer perceptron model, as depicted in figure 2.3, is an ANN that consists of an input layer and an output layer. The input layer consists of input units which receives information from the outside world. The output layer consists of perceptron units, this is where all the computations happen. Each input unit is connected to each perceptron unit via a weight which can be seen as 'the significance, or importance, of that input unit'. If a weight has a high value then the output of the perceptron will be highly influenced by that input node. This can easily be seen from what follows. The perceptron unit takes the weighted summation of the input signals and then applies an

activation function, the Heaviside step function, to this sum to produce an output $H(z)$. The weighted summation, called the net input function, is given by

$$z = \sum_j x_j W_{ij},$$

where x_j is the input signal from the j^{th} input node and W_{ij} is the weight connecting the j^{th} input node to the i^{th} perceptron. The output of the i^{th} perceptron unit is given by

$$y_i = H(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}.$$

The graph of the Heaviside step function can be seen in figure 2.4.

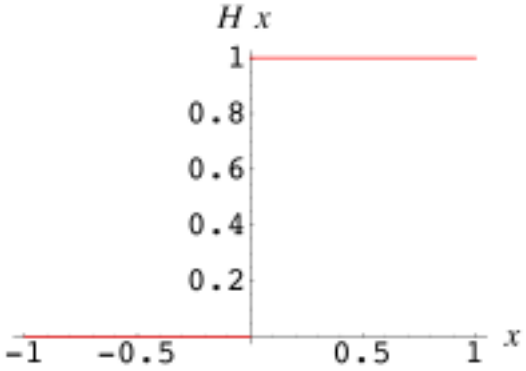

FIGURE 2.4: Heaviside step function [15]

Let us consider the task of teaching a single-layer perceptron model the difference between a football goal and a rugby goal. This would need to be done in some way that the model can understand. One way is to ask some set questions which can be answered in binary; 1 for yes and 0 for no. The questions could be 'Does the goal have a net?' and 'Is it taller than 8 feet?'. Let us say that we have a perceptron model with two input nodes, one for each question, and one output node, which outputs 0 for a football goal and 1 for a rugby goal. A typical football goal would give yes and no, which in binary would be 10 so we input 1 into the input node corresponding to the first question and input 0 into the node corresponding to the second question (in practice the input would be given by an input vector) and then expect an output of 0 to indicate it is a football goal. In other words, the idea is to train the model by showing it some number of football goals and rugby goals, 10 of each, say, and then it should be able to learn from these training samples and differentiate between the two when given unseen samples.

The question arises, how does the model learn the difference between the two goals in the first place? This is where the weights come in. They can be tuned, or updated, so that we get the desired output for each training sample. The weights represent how much of a significance each input has on classifying the goal. For example, if a question was 'Does the goal have three posts?' then this would not help the model differentiate between the two goals since typically they both have three posts. Therefore the weight corresponding to this node will be zero, or very close to zero. However the question 'Is it taller than 8 feet?' is very significant in helping the model to classify the goal since a typical football goal is 8 feet tall and a rugby goal is 52 feet tall. Hence if the answer to this question is yes, or 1, then the goal is likely to be a rugby goal. Therefore the weight corresponding to this input node will be very high since its input is of significance to the output. The weights are initialized to random values between 0 and 1.
As mentioned above, the weights need to be tuned so that at the end of the training, the model can correctly classify each goal. By calculating the difference between the actual output and the target output (the desired output), called the error, one can see whether the weights need adjusting and if they do, in which direction. The difference is given by

$$E_l^i = t_l^i - y_l^i, \tag{2.1}$$

where E_l^i, t_l^i, y_l^i is the error, target and actual output, respectively, of the l^{th} training sample for the i^{th} perceptron. It can easily be seen that if $E_l^i = 0$ then the actual output is correct and the weights do not need adjusting. If $E_l^i = -1$ then the target value was 0 but the model gave an output of 1 and so the associated weights need decreasing by a fraction of the associated input node. The opposite holds if the error was +1. Once we know if the weights need changing we can apply the following formula to update the weights

$$W_{ij}^{new} = W_{ij}^{old} + \alpha(E_l^i)x_j, \tag{2.2}$$

where W_{ij}^{new} is the updated weight, W_{ij}^{old} is the old weight, E_l^i is as defined in equation (2.1), x_j is the input node associated with the weight and α is the learning rate. The learning rate defines how fast the model should learn. In other words, it defines how drastic the weight changes should be. For a small alpha, the weights change is rather small, so the model learns slowly but has a better chance of converging to an optimal solution for the weights. For larger alphas, the weight changes are much bigger, so the

model learns a lot faster however the weights are likely to oscillate around their optimal solution. The optimal value of α is usually found by trial and error.
The weight changes need to be applied to each weight in the network for each training sample in the training set. A single pass through the entire training set is called an epoch. Eventually, after a certain number of epochs the weight changes will be zero and the model will have converged to an optimal solution and the model should now be ready to generalize. In other words, it should correctly identify unseen samples using the decision boundary it has learned.

Using the classifying goals example, consider the scenario where we have a training sample where we have a football goal that has no net, so the input vector is $[0, 0]^T$. Then by equation (2.2), it can be seen that the weight would not be updated since we would be multiplying the error by the zero input vector. To get around such complications, a bias term is introduced. This bias term is an extra input node that is set to be a constant value, usually ± 1, which then ensures that in such situations as the one described above the weights still get updated if necessary. From now on it can be assumed that the bias term is included as a node in the input layer.

The difficulty with the single-layer perceptron model is that it can not be used to solve anything more than a simple linearly separable problem. This is where multi-layer Networks become useful.

2.3.2 Multi-Layer Networks

Multi-layer networks can be used to solve non-linearly separable problems. This is because they use non-linear activation functions enabling them to learn non-linear hidden patterns in the data. In addition, multi-layer networks also have extra layers called 'hidden layers'. A hidden layer is simply a layer in-between the input and output layers containing artificial neurons, 'hidden' from the outside world. These hidden layers also allow the model to extract deep features from the data since we can apply different activation functions in each layer.
There are various different types of multi-layer networks:

1. Feed-forward neural network: This is the most simple architecture. It has an input layer, an output layer and some number of hidden layers. The data simply moves

from the input layer, through the hidden layers to the output layer. There are no feedback loops. This architecture is a popular choice for pattern recognition. Figure 2.5 depicts a typical feed forward network with a single hidden layer.

2. Recurrent neural networks: This is any neural network with feedback loops. This means data can move in both directions through the architecture so that information from later processing layers can flow to earlier processing stages. This architecture is suited to handwriting and speech recognition tasks. Figure 2.6 depicts a typical recurrent neural network with a single hidden layer and some feedback loops.

3. Radial bias function network: As the name suggests, this network uses the Radial Bias Functions (RBFs) as its activation functions. In the first hidden layer, the input vector is used as input for all the RBFs (one for each neuron in the hidden layer). The output is then a linear combination of the outputs from the RBFs. This architecture is suited to tasks such as function approximation and time-series prediction. Figure 2.6 depicts a typical radial bias function network with a single hidden layer.

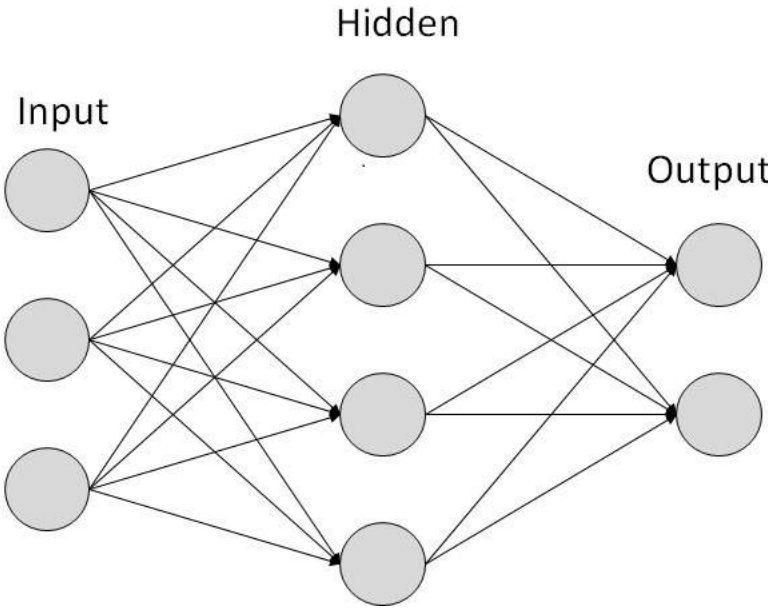

FIGURE 2.5: A feed-forward neural network [16]

ANNs can be used to solve classification and regression problems.
We have looked at a simple classification problem in the example above when we tried to classify whether an input was a football or rugby goal.

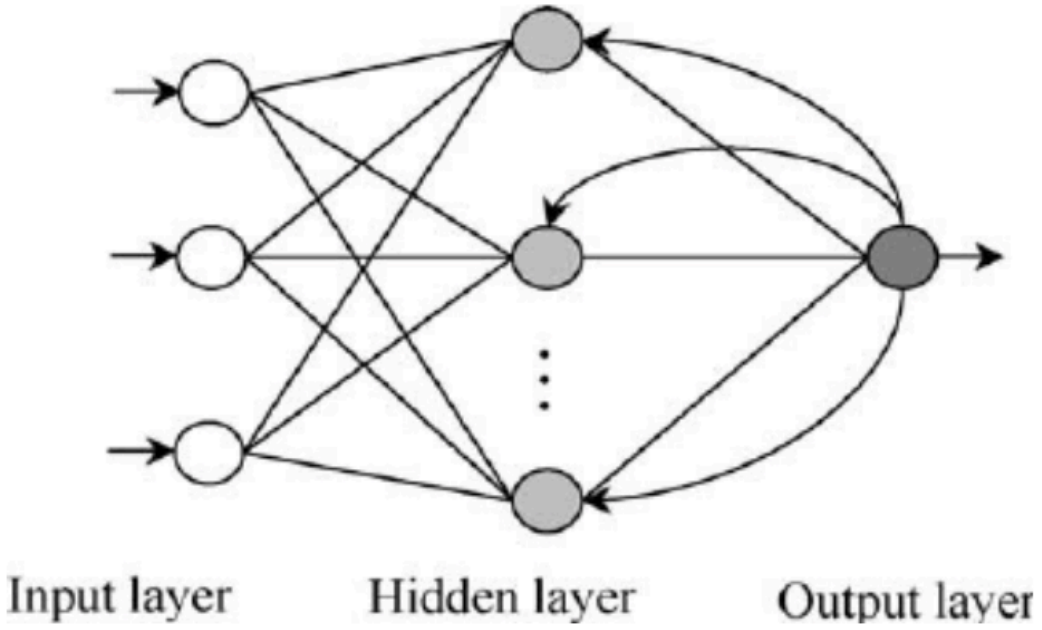

FIGURE 2.6: A recurrent neural network [17]

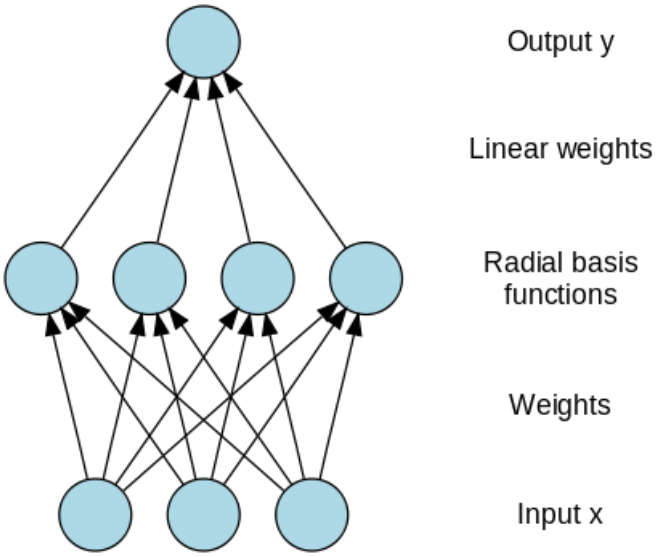

FIGURE 2.7: A radial bias function network [18]

A regression problem on the other hand is essentially a form of function approximation. The network outputs a real number which corresponds to the relationship between the input variables. For example, we could implement an ANN that will try to predict a persons height by the the length of their hands and feet

We shall focus our efforts on the feed-forward network, a simple yet powerful architecture. In particular, we will look at using feed forward networks to solve regression problems.

The basic idea of feed-forward networks is that each neuron, or node, in the input layer represents a feature in the dataset. Each of these nodes are connected to the all the nodes in the first hidden layer which are in turn connected to all the nodes in the second hidden layer, continuing to the last hidden layer, where all the nodes are connected to all the output node(s). Just like in the single-layer perceptron model, each of the connections in the architecture have a weight associated with them corresponding to the importance of that input.

As mentioned above, this network also has a non-linear activation function. There are various activation functions that can be chosen, each of which have their own merits. Two popular activation functions are:

1. Sigmoid Function: $f(x) = \frac{1}{1+\exp(-x)}$. The range of this function is between 0 and 1. A downside of using this activation function is that it has slow convergence. Figure 2.8 shows a graph of the Sigmoid function.

2. Hyperbolic Tangent Function (tanh): $f(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$. The range is between -1 and 1. Empirical evidence shows that this function enables the model to learn faster and is a popular choice for the activation function of the hidden layer. Figure 2.9 shows a graph of the hyperbolic tangent function.

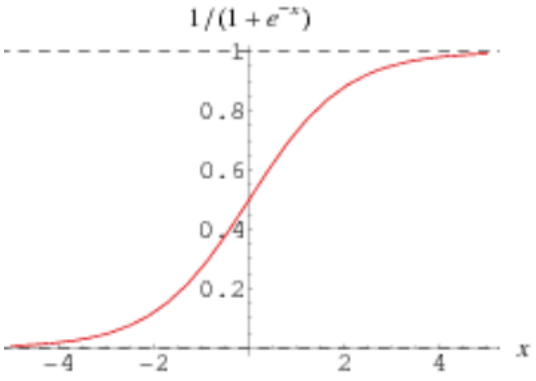

FIGURE 2.8: A graph of the sigmoid function [19]

By using non-linear activations our model has the ability to learn complex non-linear relationships in the input data, hence enabling the creation of non-linear mappings from the inputs to outputs. The functions listed are also differentiable which is an important criterion for the model to learn, which will be discussed in more detail later.

The feed-forward network learns in two stages. The first stage is the feed-forward part. Given a training sample, we first calculate the activation of the each of the n hidden

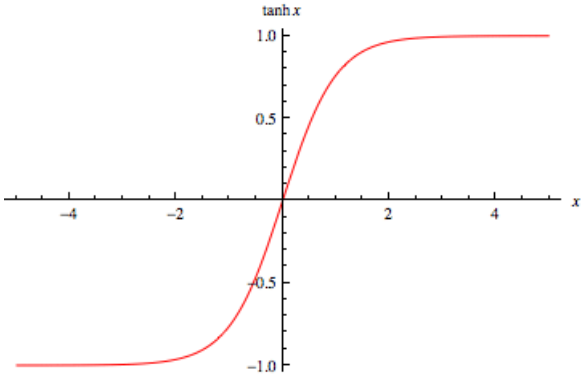

FIGURE 2.9: A graph of the hyperbolic tangent function [20]

layers, which is done just like in the single-layer perceptron model except now we are using the sigmoid function (for simplicity) instead of the heaviside step function and the input for each layer is the activation, or output, of the previous layer. For the first hidden layer, the input is simply the input vector from the input layer. This can be summarized by the formula

$$y_i^l = \sigma(z^l),$$

where y_i^l is the activation of the i^{th} neuron in the l^{th} layer and σ is the sigmoid function, given by

$$\sigma(x) = \frac{1}{1 + \exp(-x)},$$

and z^l is the sum of the net input to the neurons in layer l, given by

$$z^l = \sum_j y_j^{l-1} W_{ij}^l,$$

where y_j^{l-1} is the activation of the j^{th} node of the previous layer and W_{ij}^l is the weight connecting the j^{th} node in the $(l-1)^{th}$ layer to the i^{th} node in the l^{th} layer. It is important to pay particular attention to the order of the indexing.

As we feed-forward through the network we eventually arrive at the output node(s) which is given by

$$y_i^m = \sigma(z),$$

where y_i^m is the activation of the i^{th} neuron in the output layer for the m^{th} training sample, σ is the sigmoid function as before and z is the net input to the neurons in the

output layer, given by

$$z = \sum_j y_j^n W_{ij}^{n+1},$$

where y_j^n is the activation of the j^{th} neuron in the n^{th} (last) hidden layer and W_{ij}^{n+1} are the weights connecting the j^{th} node in the last hidden layer to the i^{th} node in the $(n+1)^{th}$ layer (the output layer).

Next we require the error between the target output and the output the model gave. Instead of simply calculating the difference between the target output and actual output we now use the mean squared error, given by

$$E_i^m = \frac{(t_i^m - y_i^m)^2}{2},$$

where E_i^m is the error of the i^{th} neuron in the output layer for the m^{th} training sample and t_i^m and y_i^m are the target and actual outputs, respectively, for the m^{th} training sample.

At this point, the model now needs to update its weights which is done through back propagation which is discussed in detail in the next section.

2.3.3 Back Propagation and Gradient Descent Optimization

Back propagation is a method used with gradient descent optimization to update the weights of ANNs in order to minimize the error of the outputs of a network.

2.3.3.1 Gradient Descent Optimization

Gradient Descent Optimization is method used to minimize an objective function, the cost function, with respect to its parameters, i.e. the weights. There are two main versions of gradient descent, whose trade-offs have to be considered when choosing a version:

1. Batch Gradient Descent (BGD): This computes the gradient of the cost function, with respect to the weights, for the entire training. In this case, the cost function is the sum squared error, given by $E_i^m = \frac{1}{2}\sum_j (t_i^m - y_i^m)^2$. This is the summation over all the errors for each sample in the training set. The downside to this version

is that it can take a long time since we need to calculate the gradient for the entire dataset. This method does however converge at a local minima.

2. Stochastic Gradient Descent (SGD): This computes the gradient of the cost function, with respect to the weights, for each training sample. The cost function in this case is the mean squared error, given by $E_i^m = \frac{(t_i^m - y_i^m)^2}{2}$. This is the mean of the square of the difference between the target and actual output. This method is quicker than BGD since it performs an update at a time. However this method performs updates frequently and therefore the error fluctuates around the optimum solution.

Stochastic Gradient Descent is usually the preferred choice due to its lower time complexity [21].
We minimize the cost function using

$$W_{ji}^{newL} = W_{ji}^{oldL} - \alpha \frac{\partial E_l^i}{\partial W_{ji}^{oldL}}, \tag{2.3}$$

The problem then arises of calculating the derivative of the error function. This is where we use back propagation.

The idea of back propagation is to start from the output layer and then modify the weights connecting the output layer to the previous layer, $(l-1)$, and then modify the weights connecting layer $(l-1)$ and the layer before that and so forth until we reach the input layer. In other words, for each neuron in each layer (except the input layer) we want to change the sum of the net input, z, by some amount Δz such that for each iteration of the back propagation algorithm, the error (otherwise known as the cost function) reduces. In order to reduce the error, we need to adjust each of the weights since the error depends on the weights. This can be achieved through the repeated use of the chain rule [22].

Let us define the net input of the j^{th} neuron in the l^{th} layer as the weighted sum of all activations of neurons of the previous layer, given by

$$\delta_j^l = \sum_i y_i^{l-1} W_{ji}^l, \tag{2.4}$$

The net input is the parameter for the activation function, the sigmoid function in our case, in each layer during the feed forward stage of training the network. In other words, given any hidden layer or output layer, l, the output for its j^{th} node is

$$y_j^l = f^l(\delta_j^l), \tag{2.5}$$

where f is the activation function [23].

Now, we want to know how much a change in weight affects the total error (the cost function), E. This can be written mathematically as

$$\Delta W_{ji} = \frac{\partial E}{\partial W_{ji}}.$$

However, since the error for the j^{th} neuron in the output layer is given by $E_j = \frac{(t_j - y_j)^2}{2}$, we see that the error is not directly a function of the weight. The error is however, a function of the output of the i^{th} neuron in the output layer. In addition to this we know that the output, given by equation (2.5), is a function of the net input. So we want the partial derivative of the error with respect to the weights however the error is a function of the output and the output is a function of the net input. If we look at the net input, given by equation (2.4), we see that the the net input is actually a function of the weights. To put this mathematically, we can calculate the partial derivative of the error with respect to the weights through the recursive use of the chain rule. That is

$$\Delta W_{ji} = \frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial y_j^l} \frac{\partial y_j^l}{\partial \delta_j^l} \frac{\partial \delta_j^l}{\partial W_{ji}^l}. \tag{2.6}$$

We can actually simplify each of the terms in the equation, the first term becomes

$$\frac{\partial E}{\partial y_j^l} = \frac{\partial \frac{(t_j^l - y_j)^2}{2}}{\partial y_j^l} = (y_j^l - t_j^l). \tag{2.7}$$

The second term becomes

$$\frac{\partial y_j^l}{\partial \delta_j^l} = \frac{\partial (1 + \exp(-\delta_j^l))^{-1}}{\partial \delta_j^l} = \frac{\exp(-\delta_j^l)}{(1 + \exp(-\delta_j^l))^2}. \tag{2.8}$$

Notice that equation (2.8) can be written in terms of the output function

$$\frac{\exp(-\delta_j^l)}{(1+\exp(-\delta_j^l))^2} = y_j^l(1-y_j^l). \tag{2.9}$$

Finally, the third term of equation (2.6) becomes

$$\frac{\partial \delta_j^l}{\partial W_{ji}^l} = \frac{\partial \sum_i y_i^{l-1} W_{ji}^l}{\partial W_{ji}^l} = y_i^{l-1} \tag{2.10}$$

The third term of equation (2.6) can be simplified like this because when we apply back propagation, we change one weight at a time. That is, when we calculate the partial derivative of the error, we do it with respect to a single weight at a time. Hence in the net input summation, only the term related to the weight we are considering will have a non-zero gradient.

Thus, substituting these term simplifications back into equation (2.6) we have

$$\Delta W_{ji} = (y_j^l - t_j^l) y_j^l (1-y_j^l) y_i^{l-1}. \tag{2.11}$$

Using this equation, we substitute it back into our original equation, equation (2.3)

$$W_{ji}^{newL} = W_{ji}^{oldL} - \alpha((y_j^l - t_j^l) y_j^l (1-y_j^l) y_i^{l-1}). \tag{2.12}$$

Equation (2.12) can be used to update the weights for a hidden layer neuron to the output neuron. To update the weights for an input neuron to hidden neuron, we also need to consider the fact that the outputs of hidden neurons contribute to the error of the neurons in the following layers. The idea is still the same as above except now we have that for each error that a specific weight, W_1, say, contributes to in the network, we have to sum the partial derivatives of those errors with respect to W_1.
Consider, for example, a single-layer multi-layer network consisting of an input layer a, a hidden layer b, and an output layer c. We update the weights connecting layer b and c using equation (2.2). Now, we can update the weights connecting layers a and b in a very similar fashion as we did for equation (2.6) except, instead of taking the partial derivative of the net input function with respect to the weights connecting layers b and c we take the partial derivative of the net input function with respect to the output of the hidden layer. This is because we can then use the output of the hidden layer in the

same way as we did for the output of the output layer and continue recursively using the chain rule until we can calculate the derivative of the error with respect to the weights connecting layers a and b.

Let ΔW^1_{ji} be defined as

$$\Delta W^1_{ji} = \sum_c (\frac{\partial E}{\partial y^c_j} \frac{\partial y^c_j}{\partial \delta^c_j} \frac{\partial \delta^c_j}{\partial y^b_{ji}}). \tag{2.13}$$

That is, since the weights connecting layers a and b will affect all the errors in layer c, we first sum all the partial derivatives of the errors in layer c with respect to the activations in layer b. We now want to calculate the partial derivatives of the activations in layer b with respect to the weights connecting layers a and b. Just like before, the activations of the layer b are functions of the net input which is a function of the weights connecting layers a and b. so we have

$$\Delta W^1_{ji} = (\sum_c \frac{\partial E}{\partial y^c_j} \frac{\partial y^c_j}{\partial \delta^c_j} \frac{\partial \delta^c_j}{\partial y^b_{ji}}) \frac{\partial y^b_{ji}}{\partial \delta^b_j} \frac{\partial \delta^b_j}{\partial W^a_{ji}}. \tag{2.14}$$

Just like before, each term can also be simplified to give

$$\Delta W^1_{ji} = (\sum_c (y^c_j - t^c_j) y^c_j (1 - y^c_j) W^c_{ji}) y^b_j (1 - y^b_j) x^a_j. \tag{2.15}$$

We then substitute equation (2.15) back into our original equation, equation (2.3)

$$W^{newL}_{ji} = W^{oldL}_{ji} - \alpha (\sum_c ((y^c_j - t^c_j) y^c_j (1 - y^c_j) W^c_{ji}) y^b_j (1 - y^b_j) x^a_j. \tag{2.16}$$

Thus we have that for a three layer feed forward network, to update the weights connecting the output layer to the hidden layer, we use equation (2.12). To update the weights connecting the input neuron to the hidden layer we use equation (2.16). If our network has more than three layers then equation (2.16) would be used to change the weights connecting the second last hidden layer and the hidden layer. Then to change the weights before those layers, we use the same principle; keep recursively using the chain rule until we can take the partial derivative of the error with respect to the weight we would like to adjust [24] .

Chapter 3

Literature Review

Aghababaeyan, Siddiqui & Khan [25] attempted to predict the Tehran stock market using artificial neural networks. They used and compared a linear regression model to a standard feed-forward network with back-propagation. They chose to work with the last four years worth of historical data to try and predict the major seven stock market indexes. In particular, factor analysis (a method used for data reduction, by representing a set of variables as a smaller number of variables) was used to try and predict the amount of cash flowing in and out of the business (cash forecasting). The prices of the stock throughout the day, intraday prices, are also included in the data set (assuming low and high price for each day; it is not made clear) as well as their corresponding date and time. They were able to show that the feed-forward network with back-propagation model was able to outperform the linear regressor with an accuracy of 97% which, considering this is new research, is very promising. A few questions can however be raised with this research. The intraday price of the stocks are of most importance to intraday traders, who buy and sell stocks within the day. Hence, with such initial accuracy this research clearly has the potential to become a very useful tool for intraday traders. Which raises the questions of whether this research could be extended to account for other types of trading strategies, such as Fundamental Analysis.

The research in this paper develops further by attempting to use textual data such as news articles, in addition to the numerical data already being used, to achieve a more accurate model. It is well known that external factors such as breaking news can have major effects on the stock market. For instance, the release of negative press on a

company, including their board members, employees and sponsors, can also negatively impact stock prices. Thus, the reasoning for also considering news articles is well justified. One may argue that by the time such breaking news is released, it would be too late to avoid any losses or capitalize on any potential gains, as the market price will reflect this news almost instantaneously. However, this is not the case, especially with positive news, to which the stock prices tend to react slowly but steadily. Stock prices will still not be instantaneous in reaction to negative news, yet there will be a far bigger and faster impact than with positive news [26]. For 83% of the time the model was able to correctly predict whether the stock price was going to move up or down, when a new piece of news was released. Clearly this model also has the potential to be very advantageous to a trader. An issue with this however is that although one knows whether the stock price will go up or down, does not make it clear as to whether to invest in, hold on to, or sell the stock. For example, if the value is only increasing by a minuscule percentage, it may be more advantageous for the trader to hold on to the stock instead of selling it. So the obvious question then is whether this model can be taken further to predict the actual percentage change in stock price using textual data, making the model an extremely useful aid for traders.

Khan, Alin & Hussain [27] attempted to use artificial neural networks to build a stock market prediction model that is more consistent than current traditional techniques such as fundamental analysis, technical analysis, and time series analysis. In particular their aim was to forecast the prices of stocks in the Bangladesh stock exchange market with a 'reasonable degree of accuracy'. One may question why it is necessary to develop or come up with a new or better model, to which they argue that current techniques are 'tedious, expensive and a time consuming process'. However, one can argue that whilst there is scope for improvement in terms of time and effort, traditional techniques are nonetheless trusted and reasonably successful techniques. So by combining artificial neural networks with a more traditional technique, we could potentially get the best out of the model.

Khan et al. used a simple multi-layer feed-forward network with back-propagation. The five chosen inputs were the general index, price-earnings ratio, net asset value, earnings-per-share and volume. They normalize the data between -1 and 1 and use the sigmoid function. Only five inputs were chosen on the basis that not all variables have a significant impact on the share market price. It could be argued that the network will

learn the significance of the variables so there is no detriment in including more variables. An area of their research that could be improved was the amount of training data. A maximum of 2 months worth of historical data was used to train the model. Increasing the amount of training data would likely improve the model and produce more accurate results. Another reason the model may not have made accurate predictions is that they were attempting to predict the daily price of the stocks, rather than quarterly or yearly prices. In the two months of data that they use to train and test the model, there could have been a lot of external factors impacting the pricing. These factors could have come from many different areas such as internal problems with the companies used, the market itself, the economy, or the news, to name a few. Therefore the quarterly or yearly prices may have been better choice to predict.

Overall, whilst the predictions themselves weren't very accurate lots of important and useful observations were made for anyone taking on similar research. They found that the more inputs used, the better the predictions became. Furthermore, it was observed that feeding the data into the model in date order resulted in better predictions than feeding in the data randomly. Lastly, in the event of any drastic changes to inputs, the predictions were noticeably different to the actual price. These are all observations that shall be carefully considered during the implementation of the model outlined in this paper.

Kenchannavar [28] attempts to use fundamental analysis and artificial neural networks to classify companies as 'good' or 'bad' investments based on their performance. They use a radial bias function (RBF) network. An RBF is a two-layer feed-forward network where the hidden nodes implement a set of radial bias functions such as the Gaussian functions. The output nodes then implement linear summation functions as in the multi-layer perceptron. Such networks are very good at interpolation. The input data used were important fundamental factors such as return on capital employed, debt-to-equity ratio, sales growth and dividend yield. The model outputs binary with 1 representing a 'good' investment and 0 representing a 'bad' investment.

The reasoning for Kenchannavar's paper was that most stock price predictions using neural networks had been concentrated on parameters such as opening & closing stock price for the day and high & low values of the index. Furthermore, other research had used techniques such as sentiment analysis, data mining, and statistical analysis in

prediction techniques. Thus, employing fundamental analysis along side artificial neural networks could yield some interesting results.

If it were somewhat successful, this study could provide the grounding for further research into using fundamental analysis and neural networks. Unfortunately the results for this study were not published in the article and therefore can not be discussed. However, the reasoning for the study was very insightful and an obvious step would be to go further than classifying good and bad investments and try to predict the actual prices or the price changes. It is interesting to learn that a lot of research into predicting stock prices primarily focused on parameters such as the various pricing of the stock throughout the day. It raises the question; why have fundamental factors not been considered when predicting stock prices? Stocks are effected by a huge amount of factors, so to see a lot of prediction models have not considered or overlooked these factors, particularly the fundamentals, is very intriguing. This could be an area of research that provides some promising results or results that explain why the fundamentals have been neglected.

In the 1980's, Ou & Penman [29] undertook a study to show that information in stock prices that leads future earnings is contained in financial statements. They demonstrate that variables found in financial statements can be summarized into one measure which predicts future earnings.

More recently, once of the aims of the research of Campanella et al. [30] was to assess the performance of fundamental analysis as a stock return predictor in European financial markets. This was achieved by searching for possible correlations between the variables and then using linear regression analysis to investigate the cause-effect relations between the variables. Their findings confirm that fundamental analysis has the ability to help investors choose share portfolios with the potential for generating abnormal returns. Furthermore, their study supports Ou & Penman in that fundamental analysis does capture the value of securities and that this value is not reflected in the stock price. The paper goes on further to give some idea of where the differences in the market value and the intrinsic value come from. It suggests that investors may not be using the information available to them properly or, that the more likely case is that economic agents - those that have an influence on the economy such as producers and consumers - do not always go by the advice of traditional theorists in terms of their principles and procedures. Another conclusion they were able to demonstrate was that the prediction model is

barely influenced by the overall economic cycle. Investopedia defines the economic cycle as "the natural fluctuation of the economy between periods of expansion (growth) and contraction (recession)".

The most important fundamentals to assess the future performance of a company is something that has been long debated and will continue to be debated for a long time. Lev & Thiagarian[31] identified a set of fundamentals that analysts claimed to be useful in security valuation and investigated these claims. They used twelve fundamental signals (a signal is a specific configuration of several fundamental variables) and they found that most of the identified fundamentals were relevant. In particular, on average, the fundamentals add 70% to the explanatory power of earnings with respect to excess returns. Furthermore they found that the relationship between the fundamentals and returns is strengthened when conditioned on macroeconomic variables. Macroeconomic variables, such as unemployment rates and inflation rates, are variables that are related to the economy as a whole. An interesting advance to this study would be to investigate whether it can be applied to different markets, such as the London stock exchange.

In 2000, Walker & Al-Debi'e [32] followed up Lev & Thiagarian's research by replicating the research on UK data. In particular they used a large sample of British companies. They also extended the study to allow the fundamental signals to be conditioned on industry state variables as well as macroeconomic variables. Their findings were mainly supportive of Lev & Thiagarian's research. In particular, they found three specific fundamental signals that were noticeably relevant in the UK; gross profit, distribution and administrative expenses, and labour force. These studies were undertaken in 1993 and 2000 respectively. The question then arises of whether these studies still relevant in today's markets. Considering the markets have significantly changed over the past 17 years, it would be of use to investigate how much of an impact those same fundamental signals have today.

Chapter 4

Problem Specification

It can be seen from the literature review that whilst there have been many attempts to use ANNs predict the prices of stocks on the stock markets, these attempts almost always overlook the fundamental factors. These attempts tend to take a range of approaches to the problem by using historical data such as stock price daily highs & lows, the opening & closing prices of stocks and even the news. This raises an interesting questions as to why using artificial neural networks with the fundamental factors to predict the stock market has not been attempted.

ANNs have the ability to learn and model complex non-linear relationships. After learning the non-linear relationships between the inputs, they can then begin to generalize and predict on unseen data. As discussed in the literature review, Ou & Penman found that fundamental analysis does capture the value of securities and that this value is not reflected in the stock price. This was then backed up by Campanella et al. Therefore, combining the capability of ANNs to learn complex non-linear relationships with the ability of fundamental analysis to capture the value of securities sounds an obvious choice in aiding a trader to build the best possible portfolios. One could use historical data to train the model so that it learns the relationships between the fundamental factors and the stock performance and then use present data to exploit these relationships to maximize returns.

This then lead to the hypothesis of this work; artificial neural networks can be used to model the non-linear relationships between the fundamental factors and the performance

of a financial portfolio. Once this hypothesis has been investigated, it is then imperative to compare the performance of the ANN against other common machine learning algorithms such as support vector machines and linear regression models. By doing so, we can verify whether ANNs were the best choice model to use or not.

Chapter 5

Data Preprocessing

The data was sourced from the Thomson Reuters Datastream service. The dataset contained the last 25 years quarterly data values from financial statements for the companies in the FTSE 350. A list of the 18 features in the dataset with definitions can be found in appendix A.

The dataset was spread out over 8 different Microsoft Excel files with each file containing data values for certain features for all 350 companies. The data was for the time period between quarter 4 1991 to quarter 4 2016. This straight away made the data difficult to work with, since the data for any single company was spread out over 8 different files and a file had up to 1400 columns. It therefore made sense to restructure the data into 350 files, where each file contains the data for a single company. The most logical way to achieve this was by using Visual Basics for Applications (VBA), the programming language for Microsoft Excel.

There were however many other issues that needed to be overcome with the data before. Firstly, the data was for all the companies in the FTSE 350 during Q4 2016. This meant that many of the companies in the dataset did not have data values for the entire time period since they were not in the FTSE for the entire time. Also, some companies did not have any entries for certain features. This made those companies data problematic to work with since machine learning algorithms do not usually work properly with data that has missing values. It was decided that the best course of action was to simply not use these companies in the dataset. This meant that when it came to training the ANN, having to find companies with the right data was no longer an issue. Some companies

had the vast majority of data values present except for a few random missing data entries. For example, for a single company all the data entries for a feature would be present except the entry for quarter 2 2012. These data values were imputed using the k-nearest neighbor imputation technique, i.e. take the mean of the k-nearest neighbours as the missing value, where k was chosen to be 2. In order to impute random missing values and also delete companies with inadequate amounts of data, two VBA macros were written. To delete companies, the macro went through each of the eight files, and for each column if there were more than four cells in a row that were empty, the word 'Delete' was entered into those cells. This was done instead of directly deleting the columns since it made splitting the data into companies a lot easier. Furthermore in order to have 'Delete' in the cell, four cells in a row had to be empty because some companies had more than one random missing entry in a row and it was decided that more than four entries in a row should not be imputed as it would alter the data too much. To impute the missing values, another VBA macro was written that found the missing values in each of the eight files and then calculated the mean of the previous and next entry.

The data could now be split into companies. Firstly, a template file was created containing the dates and column headings for each feature. Furthermore, formulas were inserted into specific columns in order to derive new features from the existing features. These added features are discussed below. The next step was to write a VBA macro for each of the 8 original files, that will split the data into companies, copy the data for a company into the relevant columns of the template file and then save the new file as the company name in a Comma Separated Value file format.

It was then necessary to create a list of the companies with all their data. By doing so, it makes training the model easier since we already know exactly which companies have the data we require. Furthermore the list was created instead of just deleting the companies without all their data as some companies could still be useful. This is because some of the companies will have all their data from a specific date and so the data from this date could be used when it came to testing the generalization of the model. To create the list, another macro was written that went through each of the company files, and if the company had all their data, the name of that company was added to a list in a separate file. The list contained 70 companies, each of which had all their data from the last 25 years. Hence in total, the final dataset contained 7000 samples.

Four features were added to the dataset, they include the forecast earnings growth for the current year, next year and the year after that. Each of these are calculated by taking the estimated earnings per share for the following year and dividing by the estimated earnings per share for the year in question. The last feature added was the percentage change in return index. This was because the percentage change is more representative of the actual change. For example, if the return index changes from 3 to 33, whilst this is only increased by 30, in terms of percentage it is a 1000% increase. Whereas a return index change from 1000 to 2000 is only a 100% increase. Thus by considering the percentage change, we can easier find stocks with the better return.

The 13 features chosen to train the model are: dividend yield, price-to-book value, price-earnings ratio, price/cash flow ratio, return index, return on invested capital, operating profit margin, current ratio, quick ratio, earnings per share and the forecast earnings growths for the next 3 years. The label of the dataset was the percentage change in return index. This was chosen as the label instead of the price since the return index tracks any capital gains of the stock and assumes all cash distributions, such as the dividends, are reinvested back into the stock. This means that the change in return index is a better indicator of the performance of a stock since it also taking into account any cash distributions rather than just how much the price of the stock will change.

The range of the values for each feature varied significantly. Some features would vary between 0 and 10 whereas other features would vary between 100 and 50000. It was therefore necessary to scale the data such that each of the features were in a comparable range. This ensures that the ANN will view each of the features equally and that the weights are not at all effected by the range of the feature values. Scaling the data also helps to make training faster and reduce the chances of getting stuck in local optima. The inputs were chosen to be scaled between 0 and 1 since the activation function output was between 0 and 1. Furthermore, the Min-Max scaling method was used to scale the data. Min-Max scaling is done by the equation

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

In addition to scaling each feature individually, the features were also scaled quarter by quarter. That is, for each quarter in the dataset, each feature was scaled using only the

data values for that quarter. This was done due to the way the model will be trained, which is explained in the next chapter.

Some features had abnormally large values relative to the rest of the values in the feature range, especially the percentage changes in return index. It was decided that these outliers should not be removed since we want the network to try to find any trends that result in these abnormal values appearing and most importantly be able to predict the abnormal changes. This is because by being able to predict these abnormal changes we would have the ability to pick portfolios to maximize our returns and avoid drastic losses.

Chapter 6

The Artificial Neural Network

In this chapter, we discuss the rationale for the design choices of the neural network.

6.1 The Design

It was decided to implement a feed-forward network with back propagation. This is because, as previously discussed, such work has not been attempted before so it made sense to start with the simplest neural network. This model can then act as the building blocks for any potential further research.

The dataset has 13 input features therefore the input layer has 13 input nodes.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output of real numbers can be approximated arbitrarily close by multi-layer feed-forward networks with just one hidden layer. Thus the model has a single hidden layer.

Since the network is a regressor which will try to predict the percentage change in return index, we simply include a single node in the output layer.

The universal approximation theorem holds for a variety of activation functions including the sigmoid function and the hyperbolic tangent function. It was decided that the hidden layer will use the hyperbolic tangent activation function due to its ability to learn faster and the output layer will use the sigmoid activation function since it is a common choice for the output layer.

The model will use the back propagation algorithm with stochastic gradient descent

hence the cost function is the mean squared error. Furthermore, by using the mean squared error, we can equally penalize both the model's underestimates and overestimates.

6.2 Hyper-Parameter Optimization

There are no definitive guidelines to setting hyper-parameters, such as the learning rate, for ANNs. The parameters depend on the problem being solved and there are different techniques to finding the best ones. Bergstra and Bengio [33] show that random search for hyper-parameter optimization generally works better than other methods such as manual search and grid search. Therefore a random search was used to choose the number of hidden nodes in the hidden layer, the learning rate and the number of epochs. That is, identify ranges that the hyper-parameters should be in, run the model with randomly chosen values within those ranges to find the configuration of parameters that give the best results. Each time the model was run, it output a graph of the average of the mean squared errors (the average cost function) for each training round. The goal was to find a configuration such that the average of the mean squared errors converged towards zero as fast and as steadily as possible. Some of these graphs can be found in the git repository (see appendix B.1).
The chosen parameters were 5 nodes in the hidden layer, a learning rate of 7.5×10^{-6} and 1000 epochs.

6.3 Running the Model

In order to train the model it was decided that the model should use an online machine learning approach. That is, the model should be trained quarter by quarter and the model should try to predict the percentage change in return index for each sample in the next quarter before being trained on that quarter. For example, the model is trained on quarter 4 1991 (the very first quarter in the dataset) and is then used to predict quarter 1 1992 (the next quarter in the dataset). The model is then trained further, or updated, (using the weights learned from the first training round) using the data from quarter 1 1992 before being used to predict quarter 2 1992, and so forth. Each time the model trains on a quarter, we shall refer to it as a training round. The

reasoning behind the online machine learning approach was that over the last 25 years, the stock market will have changed. As a result some features may have become more or less significant in terms of their relationships for predicting the percentage change in return index. We therefore want the model to be able to account for these changes.
It was mentioned in the previous chapter that the features were scaled quarter by quarter. Since the data is fed into the model in batches, where each batch is all the samples from a single quarter, we did not want the scaling to have any sort of impact on the model. That is, if all the data for a single feature was to be scaled together, the scaling for each quarter would be effected by the other quarters. This then results in the model not being able to train on each quarter independently of future quarters since the data values are not independent of each other.

Once the model has completely trained on the entire dataset, it will have also predicted the percentage change in return index for each quarter, except for quarter 1 1991 (since this was the first training quarter). For each prediction set there are 70 predictions (since there are 70 companies in the dataset) predicting the percentage change in return index for each company. This prediction set is ordered by size and then split into quintiles. That is, the prediction set is ordered by the largest predicted changes to the smallest predicted changes and then split into five equal groups where the first quintile is the top portfolio which represents the 14 largest changes and the fifth quintile is the bottom portfolio representing the 14 smallest predicted changes. The top portfolio then represents the best predicted performing companies for that quarter and the bottom portfolio represents the worst predicted performing companies for that quarter. The end result is that we have 5 virtual portfolios for each quarter. We then take the mean of the actual performance of each portfolio for each quarter and then using all the mean performances for each portfolio, we create 5 new portfolios representing the mean performance for each portfolio in each quarter. That is, the first portfolio represents the mean performance of the top predicted portfolio for each quarter.

Chapter 7

Results & Discussion

7.1 Results

In this chapter we evaluate the results of the three models; the artificial neural network, the support vector machine and the linear regressor. We then look at ways in which we can further improve the neural network and our results.

7.1.1 The Artificial Neural Network

Figure 7.1 shows the predicted and actual performances of the best portfolio. The best portfolio being the set of 14 companies that were predicted to have the greatest percentage changes in return index in each quarter. As we can clearly see, the actual predictions themselves are not very accurate and this is confirmed by the accuracy of the model only being 1% - calculated using the TensorFlow accuracy library. The predictions especially struggle to predict the bigger more drastic changes. For example, in the 3^{rd} quarter the average return index of the portfolio drops by about 21% however the model predicts an increase of about 8%. Likewise, in the 66^{th} quarter the portfolios return index dropped by over 40% however the model only predicted a drop of half of that.
Whilst the predictions are not accurate, the model however, with the exception of a few quarters, does correctly predict whether the portfolio will produce a greater or smaller return in the next quarter than in the current quarter. This can be seen by following the direction of the line for the predicted and actual performances. From about the 14^{th} quarter, the model starts to correctly predict whether the the portfolio will do better or

worse in the next quarter. It is likely that this starts from about the 14^{th} quarter due to the way the model was trained, since, before this the model will not have been fed enough data to be able to make such predictions.

These results also hold for the rest of the predicted performances of the portfolios against their actual performances. The graphs of which, can be found in the git repository (see appendix B.1).

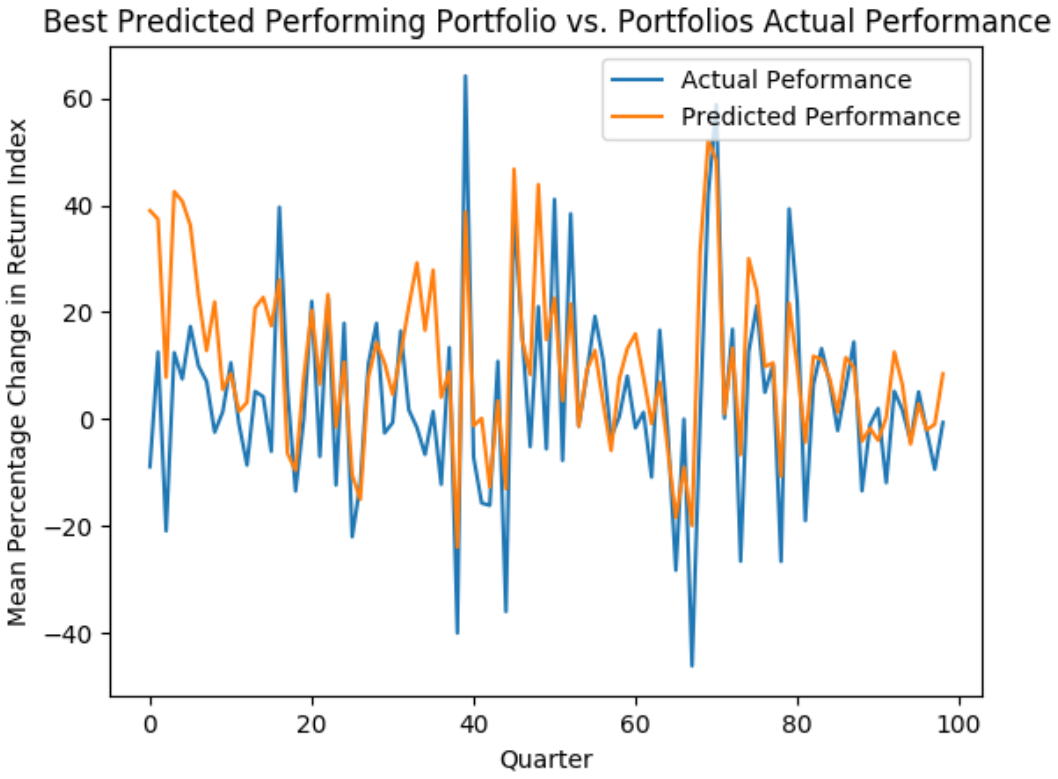

FIGURE 7.1: The best portfolios predicted performance vs. its actual performance - neural network

We have established that the models predictions are not accurate however the model does seem to be able to predict whether the portfolio will be perform better or worse in the next quarter than the current quarter. This does seem to suggest that the model is learning some sort of underlying relationship between the fundamentals and the portfolios performance. One way to verify this is by taking the actual performances of the best and worst predicted performing portfolios and conducting a Wilcoxon signed-rank test. The Wilcoxon signed-rank test is a non-parametric test that determines whether the difference between two sets of data are statistically significant. That is, the test will tell us whether the difference between the two sets has occurred by chance or not. The Wilcoxon test gives us a p-value, which signifies the significance of our results. If the

p-value is less than 0.05 than the two datasets are statistically significant. If the p-value is greater than 0.05 then the two datasets are not statistically significant and a p-value very close to 0.05 is considered marginal.

Figure 7.2 shows the actual performances of the best and worst predicted portfolios. Whilst there does not seem to be any real visible difference between the performances of the two portfolios, we are interested in whether the differences between the data are statistically significant. On conducting the Wilcoxon signed-rank test, we get a p-value of 9.57×10^{-7}. Since the p-value is much less than 0.05 we have that we can reject the null-hypothesis. That is, the hypothesis that there is no significant difference between the two datasets is false. More simply, the p-value tells us that there is a $9.57 \times 10^{-5}\%$ chance that the difference between the two datasets has occurred by chance. In addition to this, the median value of the differences between the performances of the two portfolios is 3.02% which, since it is positive, tells us that the top portfolio outperforms the bottom portfolio. More interestingly, on studying the actual percentage returns for the two portfolios we see that our top portfolio gave us an average annual return of 17% whereas our bottom portfolio gave us an average annual return of 5%, a difference of over three times. The average difference between the two portfolios is 2.88% for each quarter which means that on average we can increase our returns by 12% each year by choosing the best predicted portfolio. Thus, whilst the predictions are not accurate, the artificial neural network has successfully modelled the nonlinear relationship between the fundamental factors and the performance of a portfolio.
The calculations for these results can be found in the git repository (see appendix B.1).

7.1.2 The Support Vector Machine

We compare the results of the neural network to a support vector machine (SVM), which has the same objective; to model the nonlinear relationship between the fundamental factors and the performance of a portfolio. In order to make fair comparisons, the SVM uses the sigmoid kernel (since the ANN uses a sigmoid output activation function) and the SVM also uses the data and the results in exactly the same way as the ANN.

Figure 7.3 shows the actual performances of the top and bottom portfolios given by the SVM. The analysis of this is very similar to that of the ANN. We have that the model was not able to predict the actual changes very accurately, however, the predictions of

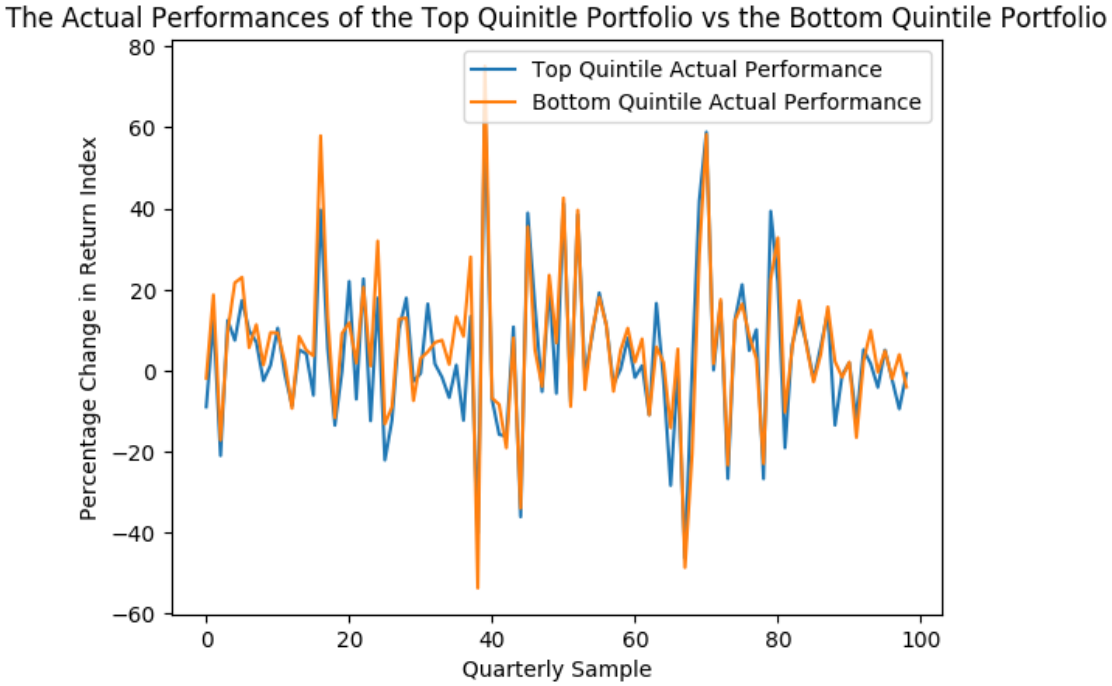

FIGURE 7.2: The best portfolios actual performance vs. the worst portfolios actual performance - neural network

the SVM do tend to follow the actual performance of the portfolio more closely than the ANN. In addition to this, the graph also suggests that the SVM would be able to correctly predict whether the portfolio will perform better or worse in the next quarter if it was a classifier. This implies that the model has been able to learn some sort of relationship between the fundamentals and the performance of the portfolio. Again, we can verify this using the Wilcoxon signed-rank test between the actual performances of the best predicted and worst predicted portfolios.

Again, as we can see in figure 7.4 there seems to be no real visible difference between the performances of the two portfolios. However, we are interested in whether there is a statistical significance between the two portfolios. The Wilcoxon signed-rank test gives us a p-value of 1.1×10^{-3} with the median value of the differences between the actual performances of the top and bottom portfolios being 2.22%. Since the p-value is less than 0.05 we can say that the differences in portfolios is statistically significant. More importantly, the positive median value means that the top portfolio outperforms the bottom portfolio and the p-value of 1.1×10^{-3} tells us that there is only a 0.11% chance that this has occurred by chance. Thus, we can conclude that the support vector machine has indeed been able to model the nonlinear relationship between the fundamental factors and the performance of a portfolio. Furthermore, the average quarterly

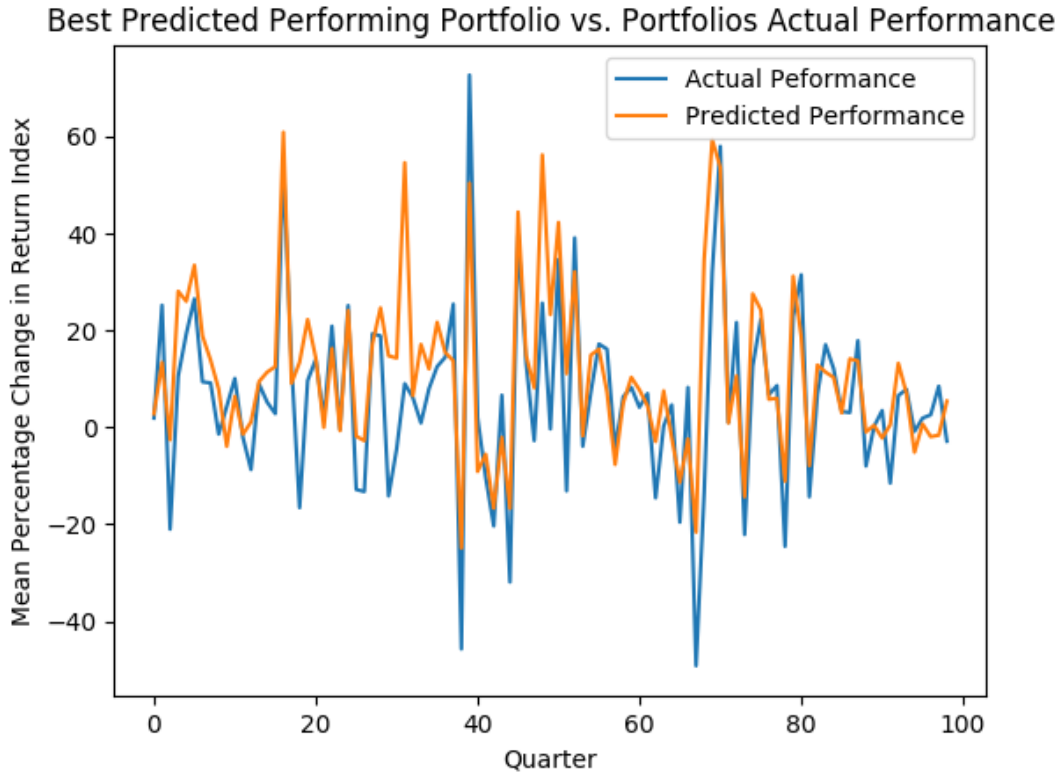

FIGURE 7.3: The best portfolios predicted performance vs. its actual performance - support vector machine

difference between the two portfolios is 2.416%. That is, our portfolio's total returns in a year could be increased by up to 8.86% by picking the top portfolio.
The ANN had a much lower p-value than the SVM, which implies that the relationship learned between the fundamentals and the portfolio performance by the ANN was much more significant than the one learned by the SVM. That is, the neural network was able to learn a more meaningful and useful relationship. Also the neural network is able to give a 3.14% greater average return than the SVM. Hence the ANN was indeed the better choice of model.

A second SVM model was also run using a radial bias function kernel and the Wilcoxon singed-rank test run in the same way. The p-value for this model 4.82×10^{-4} with the median difference being 2.169%. This again means that the SVM has indeed been able to model a nonlinear relationship between the fundamentals and the performance of a portfolio. However this time there is an even smaller chance, 0.048%, that the difference between the two portfolios has occurred by chance. This means that the nonlinear relationship learned by this model is a much more significant relationship than the one learned by the sigmoid SVM. The average difference between the portfolios has

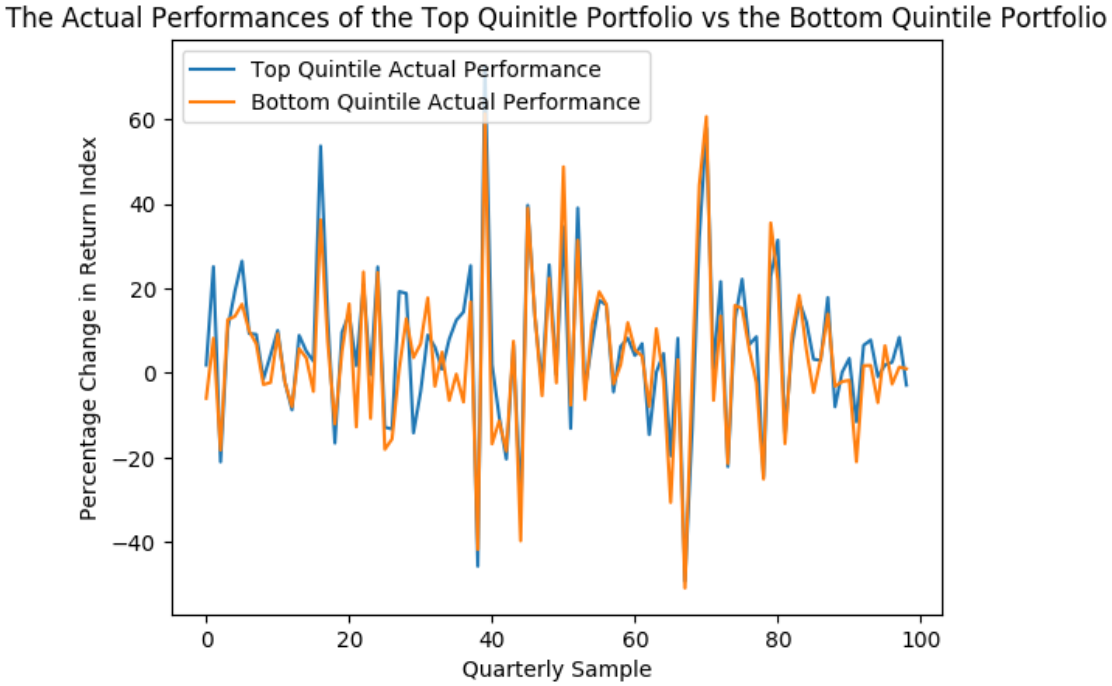

FIGURE 7.4: The best portfolios actual performance vs. the worst portfolios actual performance - support vector machine

increased to 2.167% which means that by using the radial bias function kernel instead of the sigmoid kernel, the returns on our portfolio could be increased from 8.86% to 10.89%. This result provides interesting ideas about the structure of the neural network and how it can be improved, which is discussed in the next section.

7.1.3 The Linear Regression Model

We compare the results of the neural network to a linear regression model (LRM), which has the same objective; to model the nonlinear relationship between the fundamental factors and the performance of a portfolio. In order to make fair comparisons, the LRM uses the data and the results in exactly the same way as the ANN.

The analysis of the predicted performance against the actual performance, given by the LRM, and shown in figure 7.5, is the same as that of the ANN. We have that the model fails to correctly predict the percentage changes in return index however the graph suggests that the model can predict whether the portfolio will do better or worse in the next quarter.

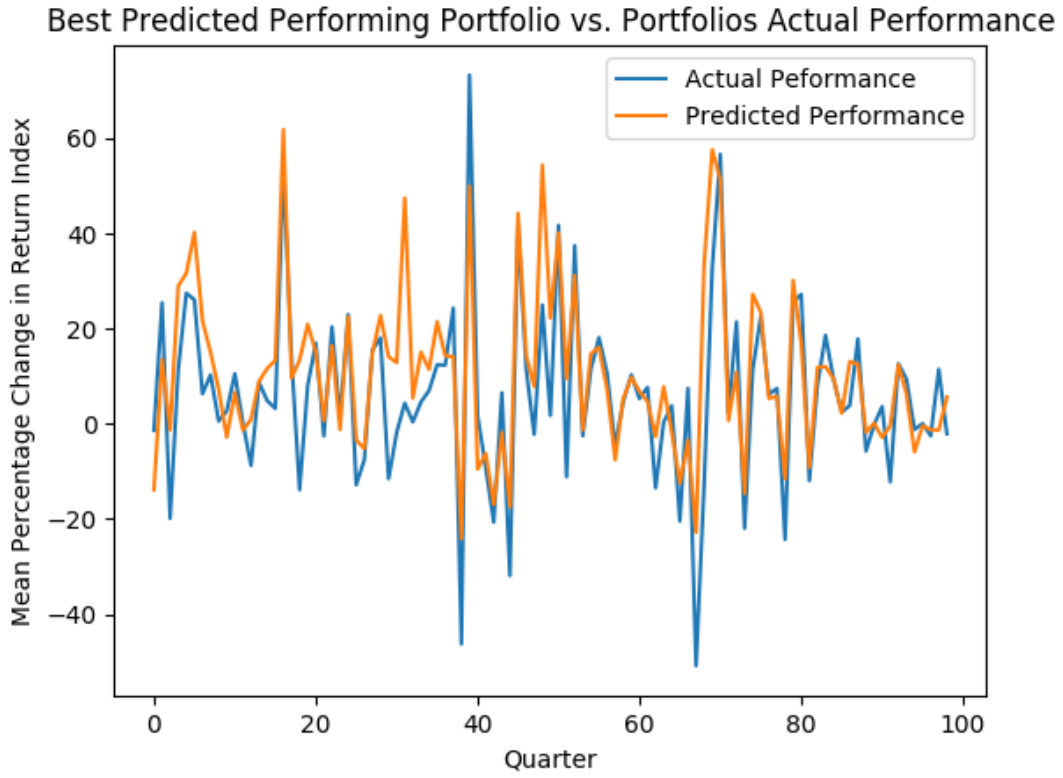

FIGURE 7.5: The best portfolios predicted performance vs. its actual performance - linear regression model

Just like with the neural network, there is no real visible difference between the top and bottom portfolios, which can be seen in figure 7.6. Conducting the Wilcoxon signed-rank test as before, we get a p-value of 4.58×10^{-4} and the median value of the differences between the portfolios is 2.56%. Hence we have that there is a statistical significance between the the two portfolios with only 0.46% chance that this difference was down to chance. Furthermore, the positive median value tells us that the top portfolio outperforms the bottom portfolio. On studying the performances of the two portfolios, we have that there is a mean quarterly difference of 2.57%. Therefore, by using a LRM to build a portfolio, we can increase our annual return by 10.68% by picking the top portfolio. Thus we can conclude that the linear regression model can be used to model the non-linear relationship between the fundamentals and the performance of a portfolio. In comparison to the neural network, the linear regressor has a higher p-value, meaning that the relationships it has learned are not as meaningful or useful as the ones learned by the neural network. In addition to this, the neural network return is able to give a 1.32% greater average return than the LRM. Hence, the neural network was indeed the better choice of model to use than linear regression model.

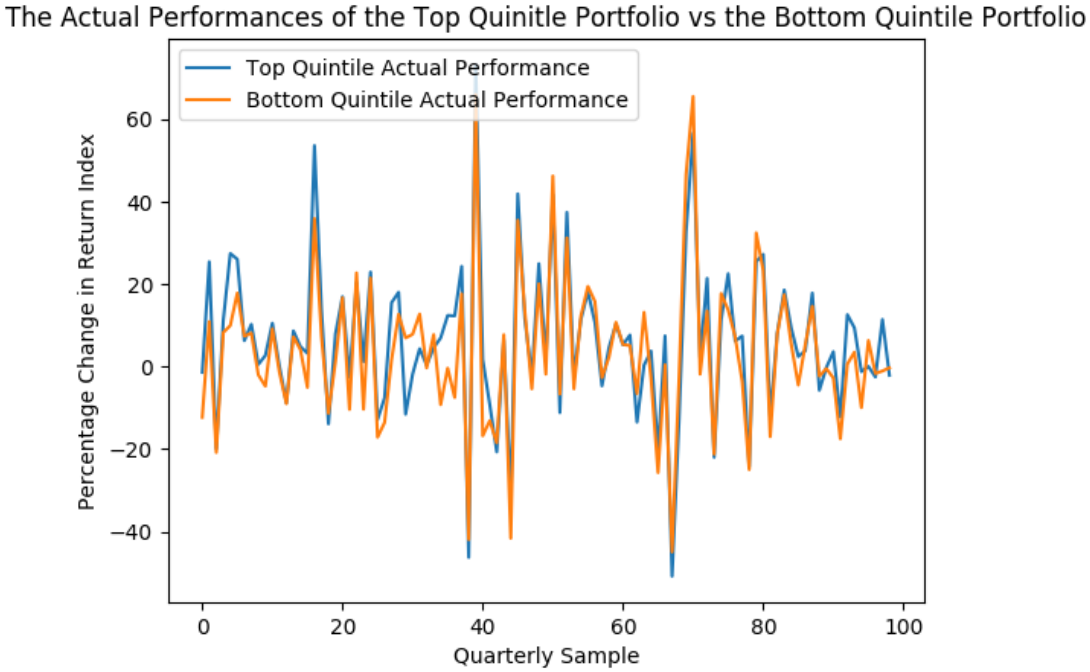

FIGURE 7.6: The best portfolios actual performance vs. the worst portfolios actual performance - linear regression model

7.2 Discussion

As discussed in the previous section, we found that there was a statistical significance between the top and bottom portfolios, with the top portfolio producing a 12% greater annual return. This statistical significance tells us that the neural network was able to model the non-linear relationship between the fundamental factors and the performance of a portfolio. However, there is definitely still scope for the neural network to be greatly improved given more time, or if this project was to be developed further.

The first improvement to the neural network is improving the significance of the relationships it has learned. That is, we try to obtain an ever lower p-value than the one we have (9.57×10^{-7}) with the goal that this can result in the top portfolio producing an even greater return than the bottom portfolio.

Firstly, the range of random hyper-parameters tested for this project could be expanded in order to find a more optimal configuration of parameters. It is possible that there are many untested parameters that instantly improve the network and its ability to learn relationships, hence, a more thorough random search in the hyper-parameter optimization is one way of improving the model.

There has also been much research into automating the hyper-parameter optimization

process in order to get the model to perform as well as possible. Adams et al. [34] recommend using Bayesian optimization and show that it can outperform other automation processes as well as human optimization. This method could also be considered for further development of the model.

The neural network used the hyperbolic tangent activation function in the hidden layer and the sigmoid activation function in the output layer since these functions are commonly used functions in such architectures. Both of these functions suffer from the same issue; the vanishing gradient problem. The vanishing gradient problem is a difficulty that arises when training neural networks that use gradient based methods such as back propagation. In particular, this problem is caused by the use of certain activation functions such as the hyperbolic tangent and sigmoid functions. It occurs because these functions map large inputs into small ranges, such as the hyperbolic tangent function mapping inputs to $[-1, 1]$ or the sigmoid function mapping inputs to $[0, 1]$. As a result the gradients of the network's outputs are really small compared to the inputs. This means that a large change in the input will only result in a small change in the output. This makes training the neural network becomes infeasible since the gradients are too small for the weights to update enough to converge towards their optimum value. To overcome such complications, activation functions that 'squash' the inputs into a small range are avoided. Instead, activation functions such as the Rectified Linear Units (ReLu) function are recommended since they do not squash their inputs. ReLu is given by $R(x) = max(0, x)$. Thus, considering other activation functions such as ReLu in place of the hyperbolic tangent function and sigmoid function is another possible improvement that could be made to the model.

We observed with the support vector machine that using a radial bias function kernel over a sigmoid kernel greatly improved the results of the model. That is, the lower p-value and greater returns meant the model was able to learn more meaningful relationships. This suggests that by using a radial bias function neural network, which is explained in chapter 2, we may be able to improve the results of the model.

The suggested improvements have so far considered the set up of the model itself. There are issues with the data that, if corrected, could also result in a better model performance. The dataset was for the companies in the FTSE 350 as of quarter 4 2016. It is important to note that a lot of these companies had not previously been in the FTSE

350, or perhaps existed, for the past 25 years, which is the main reason why a lot of companies had missing data. It was decided that only the companies with all their data available would be used in order to make sure there were no missing values. However, this meant that a lot of data was cut out as we were only left with 70 companies and 100 samples for each company.

Another issue that the dataset faced was the survivorship bias. The survivorship bias, in this case, is a result of companies that no longer exist being excluded from studies which can cause false conclusions to be made. In particular, we have that we are only considering the companies that have managed to survive in the FTSE 350 from quarter 4 1991 to quarter 4 2016. Therefore when we pick the best and worst portfolios, in actual fact we are only picking the best and worst portfolios from 70 out of a possible 350 companies. This means that at that specific period, there could have been much better or worse companies to invest in. Hence if we had all the companies that were in the FTSE 350 at each period, the performance difference between our top and bottom portfolios could have been much bigger.

The project itself may also suffer from the look-ahead bias. This is a bias created by the use of data or information in a study that was not necessarily available during the period being analyzed. So in our case, all the information that we used for each quarter to predict which stocks to invest in in the next quarter, might not have been available at that time. This means that we may have 'cheated' in the predictions the model made, in the sense that it was based on information that might have only be known later on.

There is one big change that could be made to diminish all the data issues; a dataset that only contains the data for each quarter as the FTSE 350 was in that quarter. This means that for each quarter, we have the data for all the companies that were in the FTSE 350 in that quarter. This would greatly increase the number of companies with their full datasets hence we would have a lot more samples to train the data on. It would also get rid of the surviorship bias since we look at all the available companies for that period instead of only the ones that were successful enough to survive. Furthermore we could ensure that the data only contains features that we know were available for the period being analyzed hence also getting rid of the look-ahead bias.

Chapter 8

Conclusion

The purpose of this project was to prove the hypothesis that artificial neural networks can be used to model the non-linear relationships between the fundamental factors and the performance of a financial portfolio.

Using the quarterly financial statements from the last 25 years for the companies in the FTSE 350, we picked 13 key fundamental factors that could help us to model the portfolios performance. These fundamentals included: dividend yield, price-to-book value, price-earnings ratio, price/cash flow ratio, return index, return on invested capital, operating profit margin, current ratio, quick ratio, earnings per share and the forecast earnings growths for the next 3 years. We measured the performance of a company by the percentage change in return index from one quarter to the next. This was because the return index is a good indicator of the overall performance of stock since it accounts for any capital gains and assumes all cash distributions are reinvested back into the stock. The time period analyzed was from quarter 4 1991 to quarter 4 2016.

This data was then used to train a multi-layer feed-forward neural network with back propagation. The neural network tried to predict the percentage change in return index for each quarter except the first quarter (since this was the first set of training data). Using these predictions we constructed five virtual portfolios where the first portfolio (the top portfolio) consisted of the 14 companies that were predicted to have the highest percentage changes in return index and the fifth portfolio (the bottom portfolio) consisted of the 14 companies that were predicted to have the smallest changes for that

quarter. We then took the actual performances of these portfolios and analyzed the results.

We found that the predicted performances of the portfolios were rather far from the their actual performances. However, on conducting a Wilcoxon signed-rank test between the actual performances of the top and bottom portfolios we found that the difference between the two portfolios was statistically significant. In particular, the Wilcoxon test returned a p-value of 9.57×10^{-7}. This tells us that there is a $9.57 \times 10^{-5}\%$ chance that the difference between the top and bottom portfolios is down to chance. In addition, the median difference between the two portfolios is 3.02 which, since it is positive, tells us that the top portfolio outperforms the bottom portfolio. We also found that the average annual return of the top portfolio over the 99 quarters was 17% whereas the average annual return for the bottom portfolio was only 5%. The average quarterly difference between the two portfolios was 2.88% which means that by choosing the top portfolio we can, on average, receive a 12% greater return than by choosing the bottom portfolio. This can also be seen by the difference between the average annual returns of the two portfolios.

We then compared these results to a support vector machine (SVM) and linear regression model (LRM). We found that the difference between the portfolios produced by both of these algorithms are also statistically significant. However, both of their p-values were much higher than that of the neural network. This tells us that the relationships learned by the SVM and LRM were not as significant or meaningful as the one learned by the neural network. In addition, the neural network return was able to give a 3.14% greater average return than the SVM and a 1.32% greater average return than the LRM. These results confirmed that the artificial neural network was the best choice model to go with for the project.

The statistical significance between the top and bottom portfolio allows us to conclude that the hypothesis of this project is indeed true. That is, we can use artificial neural networks to model the non-linear relationships between the fundamental factors and the performance of a portfolio.

Appendix A

The Original Features

Abbreviation	Name	Description
DY	Dividend Yield (%)	A dividend expressed as a percentage of current share price.
EPS	Earnings per Share*	Portion of company's profit allocated to each outstanding share.
MV	Market Capitalisation (£m)	The value of a company that is traded on the stock market (total # of shares * share price).
P	Price (pence)	Price of the stock.
PTBV	Price to Book Value Ratio	Ratio used to compare a stock's market value to its book value. (share price / book value).
PE	Price to Earnings Ratio	Ratio for valuing a company that measures its current share price relative to its per-share earnings. (Share price / Earnings per Share).
PC	Price to Cash Flow Ratio	Ratio of a stock's price to its cash flow per share. (Stock price / Cash flow per share).
RI	Return Index (assumes reinvestment of dividends)	Index that tracks both the capital gains of a stock over time, and assumes that any cash distributions, such as dividends, are reinvested back into the index.
DWRE	Return on Equity (%)	Measures a corporation's profitability by revealing how much profit a company generates with the money shareholders have invested (Net Income/Shareholder's Equity).
WC08376	Return on Invested Capital (%)	Assess a company's efficiency at allocating the capital under its control to profitable investments ((Net Income – Dividends) / total capital).
WC08316	Operating Profit Margin (%)	Ratio used to measure a company's pricing strategy and operating efficiency (Operating income / Net Sales).
WC08106	Current Ratio	A liquidity ratio that measures a company's ability to pay short-term and long-term obligations (current assets / current liabilities).
WC08101	Quick Ratio	Measures a company's ability to meet its short-term obligations with its most liquid assets ((current assets – inventories) / current liabilities).
YR5GTH	Earnings Growth over Last 5 Years	Annual rate of growth of earnings from investments over last 5 years.
LTMN	Mean Analysts Estimates of Earnings Growth over Next 5-7 yrs (%)	Estimated mean annual rate of growth of earnings from investments over next 5 – 7 years
EPS1MN	Mean Analysts Estimates for Earnings One Year Ahead *	Estimated mean earnings per share for one year ahead
EPS2MN	Mean Analysts Estimates for Earnings Two Years Ahead *	Estimated mean earnings per share for two years ahead
EPS3MN	Mean Analysts Estimates for Earnings Three Years Ahead *	Estimated mean earnings per share for three years ahead

* expressed as pence per share

FIGURE A.1: A description of all the features in the original dataset

Appendix B

Formalities

B.1 File structure of .zip file

The git repository can be found at: `https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2016/dxa390.git`. The 'final' folder contains all the relevant work.

The .zip file is called 'final'. It contains a folder called 'AllCompanyData' which contains all the .csv files containing the data on each company. The 'ANN Results' folder contains all the results from the neural network, including the graphs analyzing the portfolios and a .csv file which shows hows the performances of the top and bottom portfolios were calculated. The 'LinReg Results', 'SVR Sigmid Results' and 'SVR RBF Result" folder show the same things for their respective models. The 'HP Optimization' folder shows screenshots of the graphs produced when finding the optimal parameters. The 'Original Data' folder shows all the original data.

B.2 Running the code

Note, you will need to install TensorFlow and the anaconda distribution package on the machine that this file is run from. They can be installed from `https://www.tensorflow.org/install/` and `https://docs.continuum.io/anaconda/`

The 'FinalANN.py' file is the code for the artificial neural network. It can be run in the command line by changing the directory to 'Final' folder and then typing in the

command 'python FinalANN.py'.

The 'lin_reg_13ft.py' file is the code for the linear regression model. It can be run on the local machine by changing the directory to 'Final' folder and then typing in the command 'python lin_reg_13ft.py.

The 'svr_sklearn_13ft.py file is the code for the support vector machine. It can be run on the local machine by changing the directory to 'Final' folder and then typing in the command 'python svr_sklearn_13ft.py. Note it will run using the sigmoid kernel. To run it using the rbf kernel, open the file, go to line 197 and change the kernel assignment to rbf instead of sigmoid.

B.3 Third-party code

Various third-party libraries were used throughout the code of all the models to pre-process, scale and analyze data and plot any graphs. These libraries include pandas, numpy, matplotlib, sklearn, and scipy. The artificial neural network model was created using the TensorFlow libraries which includes the implementation of the back propagation algorithm. The support vector machines and linear regressor models were both created using the sklearn library.

Bibliography

[1] Stock market, (accessed 26/08/17). URL http://www.investopedia.com/terms/s/stockmarket.asp.

[2] London stock exchange, (accessed 27/08/17). URL http://www.investopedia.com/terms/l/lse.asp.

[3] Ftse group, (accessed 27/08/17). URL http://www.londonstockexchange.com/traders-and-brokers/private-investors/private-investors/stock-markets/ftse/ftse.htm.

[4] Market index, (accessed 27/08/17). URL http://www.investopedia.com/terms/m/marketindex.asp.

[5] Guide to stock-picking strategies, (accessed 27/08/17). URL http://www.investopedia.com/university/fundamentalanalysis/fundanalysis1.asp.

[6] Guide to stock-picking strategies, (accessed 27/08/17). URL http://www.investopedia.com/university/stockpicking/.

[7] S Raschka. Predictive modeling, supervised machine learning, and pattern classification, accessed 31/08/17. URL http://sebastianraschka.com/Articles/2014_intro_supervised_learning.html#learning-algorithms-and-hyperparameter-tuning.

[8] B Stecanella. An introduction to support vector machine, accessed 31/08/17. URL https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/#prettyPhoto.

[9] Dr J Brownlee. Linear regression for machine learning, accessed 31/08/17. URL https://machinelearningmastery.com/linear-regression-for-machine-learning/.

[10] M Caudill. Neural networks primer, part i. *AI Expert*, 2(12):46–52, December 1987. ISSN 0888-3785. URL http://dl.acm.org/citation.cfm?id=38292.38295.

[11] Overview of neuron structure, (accessed 28/08/17). URL https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function.

[12] Figure 2.1, accessed 31/08/17. URL http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

[13] MH Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[14] Figure 2.3, accessed 31/08/17. URL http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

[15] Figure 2.4, accessed 07/09/17. URL http://www.wolframalpha.com/input/?i=heaviside+step+function.

[16] Figure 2.5, accessed 31/08/17. URL https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm.

[17] YM Chiang, LC Chang, and FJ Chang. Comparison of static-feedforward and dynamic-feedback neural networks for rainfall–runoff modeling. *Journal of hydrology*, 290(3):297–311, 2004.

[18] Figure 2.7, accessed 31/08/17. URL https://en.wikipedia.org/wiki/Radial_basis_function_network#/media/File:Radial_funktion_network.svg.

[19] Figure 2.8, accessed 31/08/17. URL http://mathworld.wolfram.com/SigmoidFunction.html.

[20] Figure 2.9, accessed 31/08/17. URL http://mathworld.wolfram.com/HyperbolicTangent.html.

[21] S Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[22] M Nielsen. How the backpropagation algorithm works, accessed 30/08/17. URL http://neuralnetworksanddeeplearning.com/chap2.html.

[23] S He. Back-propagation algorithm, lecture 8, lh neural computaion course. Lecture Slides, accessed 19/06/17. URL https://canvas.bham.ac.uk/courses/21798.

[24] Meeden. Derivation of back proagation. Lecture Notes, accessed 30/08/17. URL `https://www.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf`.

[25] R Aghababaeyan, T Siddiqui, and MA Khan. Forecasting the tehran stock market by artificial neural network. *International Journal of Advanced Computer Science and Applications, Special Issue on Artificial Intelligence*, 2011.

[26] J Victor. How does news affect stock prices?, 2011. URL `http://www.sharemarketschool.com/how-does-news-affect-stock-prices/`.

[27] ZH Khan, TS Alin, and Md A Hussain. Price prediction of share market using artificial neural network (ann). *International Journal of Computer Applications*, 22 (2):42–47, 2011.

[28] AH Kenchannavar and SF Rodd. Fundamental analysis of financial parameters and ranking using artificial neural network. *Bonfring International Journal of Software Engineering and Soft Computing*, 2016.

[29] JA Ou and SH Penman. Accounting measurement, price-earnings ratio, and the information content of security prices. *Journal of Accounting Research*, 27:111–144, 1989. ISSN 00218456, 1475679X. URL `http://www.jstor.org/stable/2491068`.

[30] F Campanella, M Mustilli, and E DAngelo. Efficient market hypothesis and fundamental analysis: An empirical test in the european securities market. *Review of Economics & Finance*, 6:27–42, 2016.

[31] B Lev and SR Thiagarajan. Fundamental information analysis. *Journal of Accounting research*, pages 190–215, 1993.

[32] M Al-Debie and M Walker. Fundamental information analysis: An extension and uk evidence. *The British Accounting Review*, 31(3):261–280, 1999.

[33] J Bergstra and Y Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[34] J Snoek, H Larochelle, and RP Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.