Parameter Optimization of Multilayer Perceptron Network for Stock Financial Forecasting

Gianluca Cremi

Dimitar Kazakov

May 2014

BEng Dissertation

The University of York
Department of Computer Science

Number of words = 16,495, as counted by the MS Word word count command. This includes all the *body of the report*

Regulations : http://www.cs.york.ac.uk/projects/format.php

Abstract

Predicting stock closing prices have become an intense area of research in the machine learning environment, where multilayer perceptron networks (MLP) trained with back-propagation algorithm represent the dominant technique. However, identifying the optimal values for every parameter of the network is no trivial task and often leads to time consuming experimentations.

This dissertation tries to identify if there exists values of some of the parameters of an MLP that can be identified as universally optimal. The experiment consisted in optimizing the values of the following six parameters: sample set size, number of input neurons, number of hidden neurons, number of training iterations, learning rate and momentum. The experiment was conducted using the time series of fifteen companies from the FTSE100 and WEKA, a collection of machine learning algorithms for data mining tasks.

Results showed that no value resulted optimal for all the time series. However, 0.3 represented the optimal value for the learning rate in the majority of the time series. No dominant optimal value was found for the number of training iterations and for the size of the sample set. For the rest of the parameters a dominant optimal small range of parameters was identified.

Table of Contents

List of Figures

List of Tables

1 Introduction

The idea of being able to accurately predict price movement in financial markets represents a challenging, yet fascinating idea for investors and businesses. They assume that future occurrences are based on historical data [1]. On the other hand, other experts believe that no above average trading advantages can be obtained based on present and past events and data. They believe that the market is unpredictable since the behavior of the prices of the stocks is totally random as explained by the Random Walk Hypothesis (RWH). Furthermore, this concept is consistent with the theory that the market will absorb this possibility as any other type of event or information and the market will automatically compensate for it. In fact, investors will exploit any chance of profit opportunity in such a manner that will make the price of the stock regress to a point where it will be no longer profitable [2]. This claim is supported by the Efficient Market Hypothesis (EMH) developed by professor Eugene Fama in the early 1960s. It states that "the markets incorporate all available information and prices are adjusted immediately once new information becomes available" [2].However, these are theories and until now they have not been proved.

During the years researches have debated a lot about the truthfulness of the efficient market hypothesis and many arguments have been proposed against and in favor. Yaser S. Abu-Mostafa and Amir F. Atiya [1] believe that financial markets are "somewhat" predictable. They stated that *"the existence of so many trends and the undiscounted serial correlations among fundamental events and economic figures affecting the market are two of the many evidences against the efficient market hypothesis"*. Moreover, the EHM seems to be rejected by studies that have generated more reliable evidence, some of which are mentioned by Patel and Marwala [2]. For example, studies conducted on the U.S. [3] and Japanese [4] markets have shown that stock prices movement conformed only to the weak form of the EMH. Furthermore, it has been illustrated that the behavior of security prices over a 14-year period was not completely described by the EMH [5]. The RWH has also been disproved in part. A study of 234 stocks from 8 major European stock markets indicates that these stock markets exhibited a slight departure from the RWH [6]. In addition, it has been shown that monthly prices do not follow random walks in all the stock market of the countries in the Asian-Pacific region [7]. As a consequence, the above information offers encouragement for further research into developing stock market forecasting strategies and tools.

Nowadays, the methods used to forecast financial markets can be grouped into two different categories: fundamental analysis and technical analysis. Fundamental analysis studies the news events and the economic factors that influence a certain company or a commodity. On the other hand, technical analysis is based on the analysis of price patterns, as well as other important data such as trading volume and open interest figures. Technical analysis does not utilize any external economic data or any relevant news events as opposed to fundamental analysis. It is assumed that these factors get reflected in the past price pattern, and are utilized in an indirect way [1]. Therefore, technical analysis consists in studying historical data to find correlations between particular patterns and future market direction. As a consequence, during the last decades, technical analysis has been a very popular area of research in the field of Machine Learning. Many methods have been developed in order to predict stock closing prices through technical analysis.

The most famous traditional approaches to financial forecasting have always been the autoregressive integrated moving average method (ARIMA) or the Box-Jenkins. They are statistical analysis model that uses time series data to predict future trends [8]. They assume that the time series under study are generated from linear processes; however, it has been shown that real world systems are often nonlinear. For this reason, recently, other more advanced methods have been developed that are nonlinear [9]. Artificial neural networks (ANNs) have been employed to forecast financial time series [10] [11]. They are computational models inspired by animals' central nervous systems (in particular the brain) that are capable of machine learning and pattern recognition. They are usually presented as systems of interconnected "neurons" that can compute values from inputs by feeding information through the network. Another method that is starting to gain great popularity in recent years is Genetic Algorithms (GAs) [12] [13]. They are problem-solving methods (or heuristics) that mimic the process of natural evolution. Unlike artificial neural networks, designed to function like neurons in the brain, these algorithms utilize the concepts of natural selection to determine the best solution for a problem. As an alternative to ANNs and GAs it is common to find Support Vector Machines (SVMs) applied to financial forecasting problems [14]. They are a discriminative classifier formally defined by a separating hyper-plane. In other words, given labelled training data (supervised learning), SVMs output an optimal hyper-plane, which categorizes new examples. These three different types of methods represent the main approaches to financial forecasting and it is where the majority of the research concentrates on. However, many more methods exists that are being employed in financial forecasting, such as fuzzy methods [15] and Markov models [16], just to name a few.

It is very difficult to identify the best method to employ for general financial forecasting purposes due to the large number of different algorithms and models available. Usually, there is not even a model that is better than the other in every case. Some methods will be better than others in some situations while the others will perform better under other circumstances. Every method possesses its own advantages and disadvantages. In addition to that, studies have also shown contrasting results. For example, Kim [17] and L. Cao and F. Tay [18] showed that SVMs outperformed a multi-layer perceptron (MLP) trained by the Back Propagation (BP) algorithm, which is the most popular type of ANN. Nevertheless, more recently, it has been demonstrated that MLP and SVMs have produced similar results [19] [20] and in another case MLP have even outperformed SVMs [21].

This study concentrates on ANNs and in particular on the MLP method. The latest studies, in particular Ahmed et al. [21], are in favour of MLP. Furthermore, ANNs are identified to be the dominant machine learning technique in financial forecasting [22]. In particular, this study tries to understand if there exists a particular value or a small set of values of some parameters that can be identified universally optimal. Parameter optimization is a long process because there are too many parameters to be considered. This leads to time consuming experimentations in order to find the best values and to the deterioration of the forecasting performance. Cao and Tay [18] identified the presence of a smaller number of free parameters as one of the possible causes of the better performance of SVMs over MLP. Moreover, there is a lack of studies that try to optimize more than two parameters simultaneously. Therefore, the results from the study could help other researchers improve the performance of their models and reduce the time wasted by providing some guidance for the parameters selection process.

The experiment conducted consisted in using a MLP to forecast the closing prices of the stocks of 15 different companies from the FTSE100. The MLP was trained several times using different values for the parameters. The experiment was conducted using an open source program called WEKA, which is a collection of machine learning algorithms for data mining tasks developed by the University of Waikato [23]. Another, yet secondary aim of the experiment was to assert the quality of WEKA and understand if it could be a viable option to other more popular programs used for financial forecasting purposes, such as Matlab and R.

This paper is divided in the following manner: section 2 will describe in more depth what an ANN is and how it works. Section 3 will then describe the different parameters that must be considered when using an ANN for forecasting purposes. It will also show the state of the literature regarding each single parameter. Section 4 will describe the experiment carried out. Results will then be illustrated and commented in section 5, and then you will found the conclusions and future work in sections 6 and 7.

2 An Overview of ANNs

As mentioned above, ANNs are computational models inspired by animals' central nervous systems (in particular the brain) that are capable of machine learning and pattern recognition. They are usually presented as nets of interconnected "neurons" that can compute values from inputs by feeding information through the network. They represent a very powerful tool in modern quantitative finance and have emerged as a powerful statistical modeling technique. In fact, they possess some distinguishing features that make them valuable and attractive for forecasting tasks.

- First, ANNs are nonlinear data-driven self-adaptive methods that learn from examples. Therefore, they do not require any a priori knowledge about the relationships between input and outputs variables in order to be able to perform nonlinear modeling. Thus ANNs are perfectly fitted for problems in which enough data and observations are available but whose solutions are difficult to specify because they require complex knowledge [9].
- Second, ANNs can generalize quite well. Once they have learnt underlying relations present in the input data, ANNs infer the unseen part of a population with high accuracy even when the input data set is very noisy [9].
- Third, ANNs are universal functional approximators. As, illustrated by Zhang et al. [9] it has been shown that an artificial neural network can approximate any continuous function to any desired accuracy [24] [25] [26].

2.1 Historical Background

ANNs have been used in forecasting for many years already. The first application dates back to 1964 when Hu and Root [27] used the Widrow's adaptive linear network to weather forecasting. However, the research produced limited results because a training algorithm for general multi-layer networks was not present at the time. Only in 1986, the use of ANNs for forecasting purposes received a great incentive since the back-propagation algorithm was first introduced [28]. Until that moment, forecasting was dominated by traditional statistical methods such as regression and Box-Jenkins approaches. As pointed out by Zhang et al. [9] a turning point was reached when Werbos [29] [30] first formulated the back-propagation and found that ANNs trained with back-propagation outperformed the traditional statistical methods. This discovery was further consolidated by Lapedes and Farber [31] who conducted a simulated study and concluded that ANNs can be used for modeling and forecasting nonlinear time series. After that studies started to proliferate. Tang et al. [32], Sharda and Patil [33] and Tang and Fishwick [34], among others, reported results of several forecasting comparisons between Box-Jenkins and ANN models. More recently, Ahmed et al. [21] conducted a large scale comparison study of the major machine learning models employed in financial time series forecasting which showed the superiority of ANNs. Research efforts on ANNs for forecasting are considerable and the literature is vast and growing rapidly. Maciel and Ballini highlight that, many papers have been published with the sole intention of categorizing and providing a synthesis of published researches in the area.

- Wong and Selvi [35] classified the articles by year of publication, application area, journal, various decision characteristics (problem domain, decision process phase, level of management, level of task interdependence), means of development, integration with other technologies, and major contribution.

- Zang et al. [9] surveyed articles that addressed modeling issues when ANNs are applied to forecasting. They summarized the most frequently cited advantages and disadvantages of the ANN models.
- Chatterjee et al. [36] provided an overview of the ANN system and its many different uses in the financial markets. Their work has further discussed the superiority of ANN over traditional methodologies.
- Krollner [22] et al. categorized past publications according to their research motivation, the machine learning technique used, the surveyed stock market, the forecasting time-frame, the input variables used, and the evaluation techniques employed.

2.2 Structure of ANNs

The study of artificial neural networks (ANNs) has been inspired in part by the observation that simple units, such as neurons, linked together can give birth to complex and powerful interconnected structures that are at the base of biological learning systems. Each unit gets a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output, which in some cases can be feed as an input to other units [37]. It is interesting to notice that a particular neuron is relatively simple in structure and alone has limited capabilities; nevertheless, dense networks of fully connected neurons can perform complex learning operations such as classification and pattern recognition. As the proverb says “united we stand, divided we fall”.

Many different ANNs models have been proposed since the 1980s. Probably the most popular and influential of all has been the multi-layer perceptron (MLP) [9] [21] , which is the one used in the experiment. However, readers must know that there exist other types of ANNs such as bayesian neural networks (BNN), radial basis function neural networks (RBF) and wavelet neural networks (WNN) that are often used for financial forecasting purposes too. Not only MLP represent the dominant machine learning technique in financial forecasting [22], but it has been also showed that they outperformed not only other types of ANNs but other machine learning methods too [21]. These two features played a decisive role in the selection of the type of ANN to use for the experiment.

An MLP consists of a layered, feedforward, completely connected network of artificial neurons. As can be observed in Figure 1, a MLP is usually made up of three layers of neurons. The first layer represents the input layer, whose main task is to receive the input data. The last layer, or output layer, is in charge of elaborating the solution for the problem. One or more intermediate layers called the hidden layers normally separate the input from the output layer. The neural network is completely connected, meaning that every node in a given layer is connected to every node in the next layer, although not to other nodes in the same layer. Each connection between nodes has a weight (e.g., W_{1A}) associated with it. At initialization, values ranging from zero to one are assigned to each weight in a random manner.

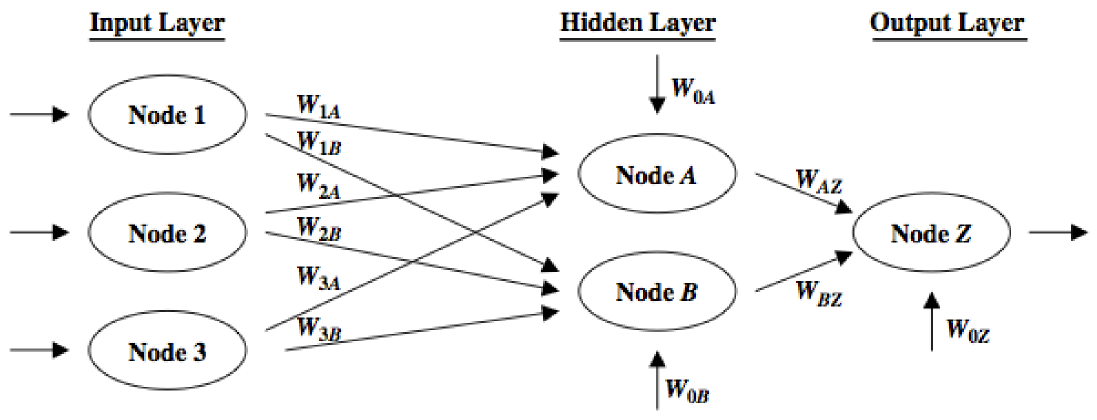

Figure 1: MLP simplified structure [38]

2.3 How a MLP works

Input neurons possess a very simple structure compare to hidden and output neurons. In fact, their job only consists in receiving inputs from the data and forwarding these values along to the hidden layer without any extra processing. Inside each hidden neuron, the inputs received and the connection weights are combined together using a combination function (usually summation, Σ) to produce a linear combination of the inputs, which we will call net, as shown in the equation below [38]. Thus, for a given node j, we have:

$$net_j = \sum_i W_{ij}x_{ij} = W_{0j}x_{0j} + W_{1j}x_{1j} + \cdots + W_{Ij}x_{Ij} \quad [38]$$

In the equation x_{ij} represents the ith input to node j, W_{ij} represents the weight associated with the ith input to node j, and there are i + 1 inputs to node j. Note that $x_1, x_2, . . . , x_I$ represent inputs from previous neurons, while x_0 represents a constant input, analogous to the constant factor in regression models, which by convention uniquely takes the value $x_{0j} = 1$. Thus, each hidden layer or output layer node j contains an "extra" input equal to a particular weight $W_{0j} x_{0j} = W_{0j}$, such as, for example, W_{0B} for node B.

The result from the combination function, in this case net_j, is then feed into an activation function. It has been noted that in biological neurons, whenever the combination of inputs to a particular neuron overcomes a particular limit the neuron starts exchanging signals with the other neurons of the network [38]. This communication behavior is not necessarily activated by a linear increase in input stimulation, thus indicating a nonlinear behavior. Consequently, ANNs imitate this conduct by using a nonlinear activation function. The most common activation function is the sigmoid function:

$$y = \frac{1}{1+ e^{-x}} \quad [38]$$

At this point the work of the hidden node is completed and the output value from the activation function y would then be passed along the connection to the output node. This process has to be completed for every neuron in the hidden layer. In the case of Figure 1, we will need to calculate the output from the activation function for node A and B. Node Z will then combines these two outputs trough a weighted sum, using the weights associated with the connections between these nodes. Note that the inputs x_i to node Z do not represent data attribute values but the outputs from the activation functions from the neurons in the hidden layer:

$$net_Z = \sum_i W_{iZ} x_{iZ} = W_{0Z}(1) + W_{AZ} x_{AZ} + W_{BZ} x_{BZ} \quad [38]$$

Finally, netZ is feed into the sigmoid activation function inside node Z. The value produce is the output from the neural network for this first pass through the network, and represents the value predicted for the target variable for the first observation [38].

2.4 Training the ANN

Before an ANN can be used to perform any desired task, it must be trained to do so. Basically, training is the procedure of identifying the best values for the arc weights because they represent the key elements of an ANN. The knowledge learned by a network is stored in the arcs and nodes in the form of arc weights and node biases [9]. It is thanks to this interconnected arcs that an artificial neural network can perform complex nonlinear mappings from its input neurons to its output neurons.

The training input data is in the form of vectors of input variables or training patterns. Corresponding to each element in an input vector is an input neuron in the network input layer. Therefore, the number of input neurons is equal to the number of input variable for each observation. As explained before in section 2.3, these input variables are feed into the network and after some processing and output value is generated by the output neuron. An error (actual – output) is then calculated by comparing the output value to the actual value of target variable for this training set observation. Often, most neural networks model use an average measurement (usually the sum of squared errors) in order to assess how well the output predictions matched the actual targets.

$$SSE = \sum_{records} \sum_{output\ nodes} (actual - output)^2 \quad [38]$$

The squared prediction errors are summed over all the output nodes and over all the records in the training set. It now becomes clear that the goal of training becomes building a set of weights that will minimize the SSE. In this way, the weights are analogous to the parameters of a regression model. The “true” values for the weights that will minimize SSE are unknown, and our task is to estimate them, given the data.

2.5 Gradient Descent Method

At this point, in order to find the set of weights that will minimize SSE, it seems logical to employ optimization methods, of which gradient-descent methods are the most appropriate. Suppose that we have a set (vector) of m weights w = w_0, w_1, w_2, ..., w_m in our neural network model and we wish to find the values for each of these weights that, together, minimize SSE. The gradient descent method, will calculate the direction that we should adjust the weights in order to reduce SSE. The gradient of SSE with respect to the vector of weights w is the vector derivative:

$$\Delta SSE(\boldsymbol{w}) = \left[\frac{\partial SSE}{\partial w_0}, \frac{\partial SSE}{\partial w_1}, \dots, \frac{\partial SSE}{\partial w_m}\right] \quad [38]$$

The above equation represents the vector of partial derivatives of SSE with respect to each of the weights.

Perhaps it is easier to show how gradient descent works with a graphical example. Larose

[38] gives a good explanation on the topic. For explanatory issues let us consider the case where a single weight w1 is present. Consider Figure 2, which plots the error SSE against the range of values for w1. The goal is to find values of w1 for which the SEE will be minimized. The optimal value for the weight w1 is indicated as w_1^*. It would be useful to develop a rule that would help us move our current value of w1 closer to the optimal value w_1^* basing our decision only on the current location of w1. The equation below achieves the task.

$$w_{new} = w_{current} + \Delta w_{current} \quad [38]$$

Δw$_{current}$ represents the "change in the current location of w."

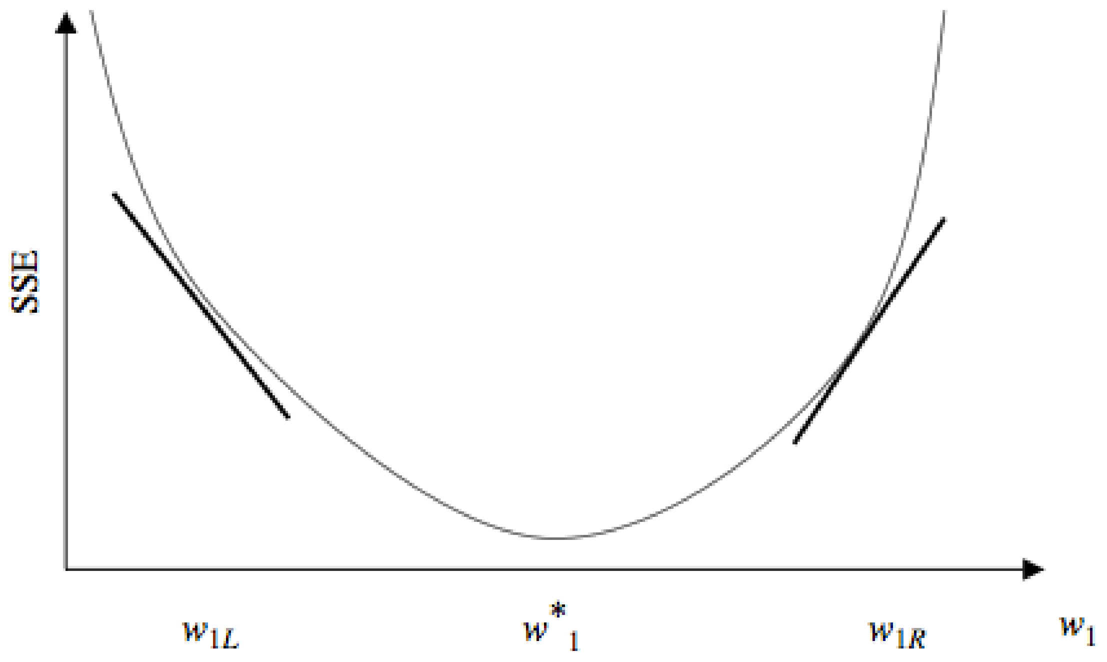

Figure 2: Gradient descent method illustration [38]

If w$_{current}$ is near w1$_L$ we should increase its value to bring it closer to the optimal value w_1^*. On the other hand, if w$_{current}$ was near w1R, we would need instead to decrease its value, to bring it closer to the desired optimal value w_1^*. Now we introduce the derivative ∂SSE/∂w1 that is simply the slope of the SSE curve at w1. For values of w1 close to w1$_L$, this slope is negative, and for values of w1 close to w1R, this slope is positive. Therefore, the direction for adjusting w$_{current}$ is the negative of the sign of the derivative of SSE at w$_{current}$, that is, −sign(∂SSE/∂w$_{current}$).

Using −sign(∂SSE/∂w$_{current}$) will always generate the correct direction to take in order to optimize the value of w$_{current}$. So, we have found a way to determine the direction, now we have to determine by how much we should move the value of w$_{current}$. For this purpose we will use the magnitude of the derivative of SSE at w$_{current}$. When the curve is steep, the adjustment will be large, since the slope is greater in magnitude at those points. When the curve is nearly flat, the adjustment will be smaller, due to less slope. Finally, the derivative is multiplied by a positive constant, called the learning rate, whose purpose will be explained in section 3.8, with values ranging between zero and one. The resulting form of Δw$_{current}$ is as follows:

$$\Delta w_{current} = -\eta\left(\frac{\partial SSE}{\partial w_{current}}\right) \quad [38]$$

In simple words this equation indicates that the change in the current weight value equals a negative small constant times the slope of the error function at wcurrent .

2.6 Back-Propagation

The back-propagation algorithm is probably the most popular ANN training algorithm [9]. It takes the prediction error (actual – output) for a particular observation in the data set and, as the name suggests, percolates the error back through the network, assigning partitioned responsibility for the error to the various connections. Hence, as illustrated in section 2.5, it uses the gradient descent method to adjust the weight on the connections to reduce the error.

Mitchell [37] derived the back-propagation rules for a network that uses the sigmoid transfer function and the gradient descent method as follows:

$$w_{ij,new} = w_{ij,current} + \Delta w_{ij} \qquad \text{where} \qquad \Delta w_{ij} = \eta \delta_j x_{ij} \quad [37]$$

Now η represents the learning rate and x_{ij} signifies the *i*th input to node *j*. The new component δ_j indicates the responsibility for a particular error belonging to node *j*. The error responsibility is calculated using the partial derivative of the sigmoid function with respect to net_j. It takes different forms depending if the selected node belongs to the hidden or to the output layer:

$$\delta_j = \begin{cases} output_j(1 - output_j)(actual_j - output_j) & for\ output\ layernodes \\ output_j(1 - output_j) \quad \sum_{downstream} W_{jk}\delta_j & for\ hidden\ layernodes \end{cases} \quad [37]$$

In the equation above $\sum_{downstream} W_{jk}\delta_j$ refers to the weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer neurons. It should be noticed that the back-propagation rule is the cause for which the attributes must to be normalized to be between zero and one. For example, suppose that input data contains very large values and these are not normalized then, the weight adjustment $\Delta w_{ij} = \eta\delta_j x_{ij}$ would be overcome by the data value x_{ij}. As a consequence the error propagation (in the form of δ_j) through the network would be overwhelmed, and learning (weight adjustment) would be suppressed.

2.7 Termination Criteria

The procedure described in section 2.6 is them repeated for every observation in the training data set, thus at every passage the weights are adjusted to reduce the forecasting error. This process may go on for large amount of time; hence a termination criterion is needed. There are different factors that can be used as a termination criterion. If training time is a concern, one may simply program the algorithm to stop executing after a certain number of iterations have been completed or a certain amount of real-time has elapsed [38]. Nevertheless, this increase in training speed is probably obtained at the expenses of the prediction accuracy of the model. Another possibility is to set some low threshold level and then train the network on the training data set until the threshold level is reached by the SEE. Although tempting, this approach is not recommended because the patterns in the training set might be memorize by the neural network thus losing generalizability to unseen data. This phenomenon is known as over-fitting and represents one of the main disadvantages of neural networks. Therefore, in order to achieve acceptable prediction errors it is important to reduce the risk of over-fitting.

Most neural network models adopt the following cross-validation termination procedure suggested by Larose [38]:

- Retain part of the original data set as a holdout validation set.
- Proceed to train the neural network as above on the remaining training data.
- Apply the weights learned from the training data on the validation data.
- Monitor *two sets of weights*, one "current" set of weights produced by the training data, and one "best" set of weights, as measured by the lowest SSE so far on the validation data.
- Terminate the algorithm, only when the current set of weights has reached a significant greater SSE than the best set of weights,

Another great problem of ANNs is that they are not guaranteed to arrive at the optimal solution, known as the "global minimum" for the SSE. In fact, the algorithm may get stuck in a local minimum, which represents a good, yet not optimal solution. In practice, there are some good solutions to this problem:

- For example, multiple networks may be trained using weights with different initial values, with the best-performing model being chosen as the "final" model.
- Researchers have also developed an improved version of the back-propagation algorithm called the stochastic version that decreases the chance of becoming stuck in a local minimum as it introduces randomness to the gradient descent method [39].
- Alternatively, a *momentum* term may be added to the back-propagation algorithm, with effects discussed in section 3.8.

3 ANN Design

Building a neural network forecaster for a particular forecasting problem is a no trivial task. Many different factors that affect the performance of an ANN must be considered carefully. One critical decision is to determine the appropriate architecture, that is, the number of layers, the number of nodes in each layer, and the number of arcs which inter-connect with the nodes. The selection of these parameters is complicated and time consuming because they are problem-dependent. This means that for every forecasting problem there exists a particular set of values for the parameters of the ANN that represents the best choice. Zhang et al. [9] illustrates some of the techniques develop to help in the identification of this values; pruning algorithm [40] [41] [42] [43] [44], the polynomial time algorithm [45], the canonical decomposition technique [46] and the network information criterion [47]. However, these procedures usually are not that useful because they are very complex and difficult to implement. Moreover, none of these methods can guarantee the optimal solution for all real forecasting problems. Nowadays, there is no exact method that allows researchers to find the optimal values for these parameters. Guidelines are either heuristic or unreliable since they were obtained through experiments with major limitations. Hence the design of an ANN is more of an art than a science [9].

Recently, as pointed out by Zhang et al. [9], genetic algorithms have gained great popularity in improving the structure of ANNs [48] [49] [50] [51]. Genetic algorithms are optimization procedures that can simulate natural selection and biological evolution to reach more efficient ANN learning process [52]. Due to their unique properties, genetic algorithms are often implemented in commercial ANN software packages [9], however they are not present in WEKA.

Apart from the network architecture, researchers must consider the selection of the type of activation functions of the hidden and output nodes, the training algorithm. They must also choose if data transformation or normalization methods are going to be used, the size training and test sets, and performance measures.

In this section we survey the literature regarding ANN design and indicate the values that we are going to use during the experiment for every single parameter. The following 8-step design methodology was used. It was created by Kaastra and Boyd [11], which in turn was inspired by the research of Deboeck [53], Masters [54], Blum [55], and Nelson and Illingworth [56]. We varied it slightly from the original version by including the number of input neurons in step 5.

- Step 1: Variable selection
- Step 2: Data collection
- Step 3: Data preprocessing
- Step 4: Training, testing and validation sets
- Step 5: Neural network paradigms
    - Number of input neurons
    - Number of hidden neurons
    - Number of output neurons
    - Transfer or activation function
- Step 6: Evaluation Criteria
- Step 7: Neural network training
    - Number of training iterations
    - Learning rate and momentum

- Step 8: implementation

3.1 Step 1: Variable Selection

In this step one must chose the type of data that is going to be feed into the artificial neural network. Thus, it is important to research and understand which input variables are relevant in the market that we want to forecast. This is no easy task since the ability of neural networks to find complex nonlinear connections between a number of apparently unrelated variables is one of the key reasons for relaying on them. Nevertheless, economic theory can help in choosing variables which are probable important predictors.

Essentially, there are two economic types of inputs that can be used; technical and fundamental inputs. Technical inputs are defined as lagged[1] values of the dependent variable[2] or indicators calculated from the lagged values. Fundamental inputs are economic variables which are believed to have an influence on the dependent variable [11]. Using lagged values of the dependent variable(s) or its first difference as inputs to the network model represents the easiest way. Ahmed et al. [21] demonstrated that using just simple lagged values of the dependent variable was more efficient than using its first difference. For this reason and because of the simplicity and popularity of the approach we have decided to use lagged values of the dependent variable in our experiment.

The frequency of the data is another factor to be considered and it depends on the objectives of the researcher. A typical off-floor trader in the stock or commodity futures markets would likely use daily data if designing a neural network as a component of an overall trading system. An investor with a longer term horizon may use weekly or monthly data as inputs to the neural network. An economist forecasting the GDP, unemployment, or other broad economic indicators would likely use monthly or quarterly data [11]. In the experiment, we used daily data since the neural network model was a one step-ahead forecaster.

3.2 Step 2: Data Collection

The research must consider cost and availability when collecting data for the variables chosen in the previous step. In fact, technical data is readily available from many vendors at a reasonable cost whereas fundamental information is more difficult to obtain. It is important that the source of the data is reliable, however, all data should still be checked for errors. For the experiment data was freely collected from Yahoo Finance [57].

In the case of days with no trading, the missing data may be required to be manually inserted. Three of the common techniques employed to contend with days with no trading [58], are:

- Use the data for trading days only while ignoring the days with no trading.
- Days with no trading can be assigned a value of 0.
- The value of days with no trading can be approximated by a model built for the purpose.

1616

[1] "Lagged" means an element of the time series in the past. For example, at time t, the values $y_{t-1}, y_{t-2}, y_{t-p}$ are said to be lagged values of the time series y.

[2] Dependent variable is the variable whose behaviour should be modelled or predicted.

We decided to use the first technique as a result from Patel and Marwala [2]. They conducted experiments utilizing techniques 1 and 2 and they found that technique 1 resulted in lower error values. Moreover, they did not create a model to determine the value for the days with no trading, as suggested by technique number 3, because it was feared that the values calculated would have contributed significantly to the final error of the network.

3.3 Step 3: Data Preprocessing

Often, data processing is crucial in order to improve prediction performances when applying artificial neural networks for financial forecasting problems. It is rare to see ANN forecaster feed with raw input data. Sometimes data preprocessing is even required in order to allow the network to work correctly. At least, the input data set must be scaled between the upper and lower bonds of the transfer functions (usually between zero and one or minus one and one) as illustrated in section 2.6. This is accomplished through data normalization, for which there exist different strategies. The most common of this consists in acquiring the minimum and maximum values within the data sets and then using the formula below:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \text{, [2]}$$

where

X_{norm} is the normalized value,

X is the actual data,

X_{min} is the smallest value within the data set,

X_{max} is the largest value within the data set.

Even if the activation function does not require the data to be contained within a specific range, normalization may still be advantageous to avoid computational problems and to facilitate network learning [33]. Furthermore, normalization can help in reducing the fluctuation and noise level within the data. There are also a variety of practical reasons which illustrate that normalizing the data sets can result in faster training and reduce the chances of obtaining local optima [59].

Zhang et al. [9] pointed out that, as a result of normalizing the data set, the observed output of the network will correspond to the normalized range. Hence, to be able to interpret the results correctly they must be rescaled to the initial range. In the same way, performance measure should only be applied to the rescaled outputs.

Another important topic in time series forecasting is seasonality and trends. Zhang and Qi [60] have found out that neural networks are not able to capture seasonal or trend variations effectively while utilizing unpreprocessed raw data. Either detrending or deseasonalization can dramatically reduce forecasting errors, thus improving the overall performance of the forecaster. Furthermore, combining detrending and deseasonalization was found to be the most effective data preprocessing approach. Although recommended, we didn't detrend nor deseasonalize the data because of the lack of knowledge in the matter and the lack of tools to perform these actions. We understand this represents a limitation to our experiment.

3.4 Step 4: Training, testing and validation sets

Two of the common problems that can arise in training the ANN are over-fitting and under fitting. Over-fitting occurs when the network loses its generalization capabilities and tends to memorize the training data. Under-fitting occurs when the network does not follow the data at all because is not able to identify the underlying relations in it [2]. Common practice is to divide the time series into three distinct sets called the training, testing and validation sets. Although they do not guarantee that over-fitting and under-fitting won't occur, they have proved to be very useful. The training data set is the largest and is used to train the ANN to find the general pattern between its inputs and outputs. The testing data set, usually ranging in size from 10% to 30% of the training set, is used to assess the ability of trained network to generalize and the validation data set is used to confirm the prediction quality of the developed networks. The size of the latter should consist of the most recent contiguous observations. Moreover, there should be a balance between obtaining a sufficient sample size to evaluate the trained network and having enough remaining observations for both training and testing [11].

The testing set can be either randomly selected from the training set or consist of a set of observations immediately following the training set. With the random strategy the risks of evaluating data against testing sets that present a trend is largely reduced. Otherwise, the testing set will favor networks which specialize either on strong uptrends or strong downtrends at the expense of networks which generalize by performing well on both. On the other hand, the advantage of using the observations following the training set as testing facts is that these are the most recent observations (excluding the validation set) which may be more important than older data [11]. Literature does not agree in identifying a superior method. We have decided to go for the second method and we have segregated the data in time order. In other words, the data of the earlier period was used for training while the data of the latest period was used for testing. We understand that this approach could lead to recency problems as explained before. Unfortunately we weren't able to perform any validation because WEKA allows the user to split the data set only into training and testing. When we realize this problem, it was already too late to change software. However, this is only a minor issue since it is also common to use one test set for both testing and validation purposes [9].

Another closely related factor is the sample size. No guidelines exists regarding the optimal value for the sample set size. The amount of data for the network training depends on the network structure, the training method, complexity of the problem and the amount of noise in the data on hand. Nam and Schaefer [61] found out that an ANN forecaster performed better as the training sample size increases. In general, as in any statistical approach, the sample size is closely related to the required accuracy of the problem. The larger the sample size, the more accurate the results will be. However, we believe that later data should have a bigger influence than earlier data in one step-ahead forecasting. Therefore, in order to verify this hypothesis we used four different sample sizes in the experiment. In particular, the sizes used were 3, 6, 13 and 25 months of daily data. Each data set was in turn divided into training (80 %) and testing (20 %) sets.

3.5 Step 5: Neural network paradigms

Many features have to be considered when building an artificial neural network model. One has to consider each single neuron first. For each one of then the appropriate activation function must be chosen. After this, one has to determine the whole structure of the network and has to consider how the neurons relate to each other. This section will address the selection of the number of input neurons, hidden layers, hidden layer neurons, and output neurons, as well as the transfer function for a multilayer perceptron network.

3.5.1 Number of input neurons

In a time series forecasting problem, the number of input nodes corresponds to the number of lagged observations used to discover the underlying pattern in a time series. The problems then becomes identifying the ideal number of lagged variables to use. Nevertheless, currently there is no suggested systematic way to determine this number. Often, it is ideal to have a network structure with few elements, thus having few lagged variables would be preferable. Nevertheless, careful consideration must be paid because too few input neurons can lead to over-fitting while too many neurons can lead to under-fitting [9]. Most authors design experiments to help select the number of input nodes while others adopt some intuitive or empirical ideas. For example, Sharda and Patil [33] and Tang et al. [32] used 12 inputs for monthly data and four for quarterly data heuristically. Unfortunately, exploring the literature, we found no consistent results regarding the identification of an optimal value.

Zhang et al. [9] believe that the number of input neurons is probably the most critical decision variable for a time series forecasting problem since it contains the important information about the complex nonlinear autocorrelation structure in the data. Considering this, we have decided to test four common input neuron numbers find in the literature, which are respectively 2, 4, 8 and 12.

3.5.2 Number of hidden layers

The number of hidden layers provides the network with its ability to generalize. In theory, a neural network with one hidden layer with a sufficient number of hidden neurons is capable of approximating any continuous function with any desired accuracy [24]. As indicated by Zhang et al. [9] most authors use only one hidden layer. However, networks with one hidden layer may require a very large number of hidden neurons, which is not optimal because it increases the training time and it deteriorates the network generalization capability. In fact, some authors have found that using two hidden layers instead of one produce more accurate predictions [62] [63]. In practice, neural networks with one and occasionally two hidden layers are widely used and have performed very well. It is not recommended to increase the number of hidden layers to much since this will increase computation time and the danger of over-fitting which leads to poor out-of-sample forecasting performance [11]. Authors have concluded that a network never needs more than two hidden layers to solve the majority of the problems, including forecasting [64] [65]. Zhang et al. [9] say that one hidden layer may be enough for most forecasting problems, thus based on their opinion we used only one hidden layer during the experiment.

3.5.3 Number of hidden neurons

The issue of determining the optimal number of hidden nodes is a crucial yet complicated one. It is in the hidden neurons that most of the processing happens. As usual it is better for the performance of the forecaster to build simple networks with fewer hidden neurons because, usually, they show better generalization capabilities and are less prone to over-fitting. On the other hand, under-fitting may arise if too few hidden nodes are implemented thus making the forecaster unable to learn from data [9]. Identifying the optimal value requires experimentation. Essentially, there exist three methods that are commonly used, these are the fixed, constructive and destructive. In the fixed approach, a group of neural networks with different numbers of hidden neurons are trained and each is evaluated on the testing set using a reasonable number of randomly selected starting weights. The network with the least error found should be selected because it is able to generalize best. This approach is time consuming, but generally works very well.

While the fixed approach is static, the other two are dynamic. In fact, both the constructive and destructive approaches involve changing the number of hidden neurons during training. More precisely, the constructive approach involves adding hidden neurons until network performance starts deteriorating, while the destructive approach is similar except that hidden neurons are removed during training [11]. Regardless of the method used, two rules of thumb exist: always select the network that perform best on the testing set, in case of a tie always go for the one with the least number of neurons.

It is interesting to notice that, although identifying the optimal number of hidden neurons is crucial and complicate, Tang and Fishwick [34] have investigated the effect of hidden neurons and have found that the number of hidden neurons does have an effect on forecast performance but the effect is not quite significant.

Many authors believe that the optimal number of hidden neurons is a function of the input neurons and several practical guidelines exist based on this hypothesis. For our experiment, we have chosen four of the most popular and reasonable practices. These include using "2n" [66], "n/2" [67], "3n" [68], where n is the number of input nodes. Finally Zhang et al. [9] noticed that several studies [33] [34] [69] have reported that networks in which the number of hidden neurons was equal to the number of input neurons achieved better forecasting results.

3.5.4 Number of output neurons

The number of output nodes is probably the easiest parameter to choose since it directly depends on the forecasting range of the study. There are two types of forecasting: one-step-ahead (which uses one output node) and multi-step-ahead forecasting. Zhang et al. [9] go into more depth and describe the two most common ways of making multi-step forecasts. However, for the purpose of this paper we do not describe them since we only deal with one-step ahead forecasting.

3.5.5 Transfer functions

Transfer functions are mathematical formulas that determine the output of a processing neuron. The majority of current neural network models use the sigmoid function.

$$y = \frac{1}{1+e^{-x}} \quad [38]$$

The sigmoid function is continuously differentiable which is a useful property for network training [11]. Furthermore, depending on the input, it combines three different types of behavior. To illustrate this fact, let's consider the sigmoid function in Figure 3. We should point out that but f(x) can theoretically take any real-valued inputs. Through much of the center of the domain of the input x (e.g., −1 < x < 1), f (x) is characterized by nonlinear behavior. As the input moves away from the center, f (x) becomes curvilinear, while near extreme values, f (x) becomes almost constant. Moderate increments in the value of x produce varying increments in the value of f (x), depending on the location of x. Near the center, moderate increments in the value of x produce moderate increments in the value of f (x); however, near the extremes, moderate increments in the value of x produce tiny increments in the value of f (x) [38]. The codomain of the sigmoid function lies between zero and one. For this reason, it is sometimes defined as squashing function since it takes any real-valued input and returns an output that belongs to this range. This also explains why data normalization is need when using neural networks for financial forecasting purposes.

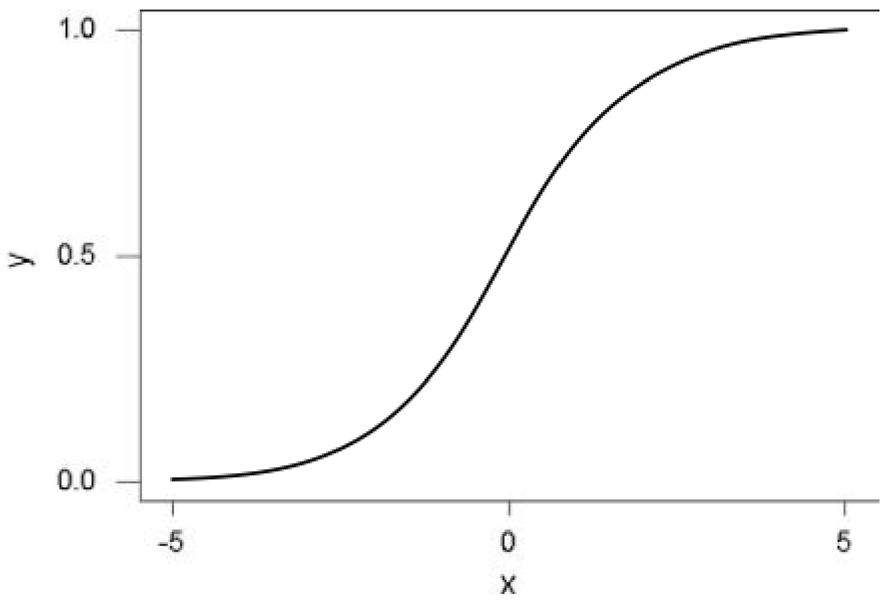

Figure 3: Graph of the sigmoid function [38]

Other activation functions such as the hyperbolic tangent, step, ramping, arc tan and linear have also been proposed. They are used in situations where the sigmoid function is not recommended. For example, Klimasauskas [70] stated that the hyperbolic tangent is more suitable in networks used to learn deviations from the average, while if learning involves average behavior the sigmoid function is still preferred. Moreover, in case binary variables are present it would be better to use the ramping and step functions since the sigmoid function approaches zero and one asymptotically [11].

During the experiment the sigmoid transfer function was used because it is the most popular and appropriate.

3.6 Step 6: Evaluation Criteria

The ultimate and the most important performance measure is certainly the prediction accuracy a neural network can attain on the testing data set. Apart from this, in some cases other measures can proved to be relevant. For example, sometimes forecasting methods can be quite demanding in computational power and time [21]. If the method takes too much time to train or the computer available is not powerful enough to run it, then it is as useless as having a method whose prediction accuracy is very low.

Another issue is that there is no suitable measure of accuracy for a given problem that is universally accepted by the forecasting community. An accuracy measure is often defined in terms of the forecasting error which is the difference between the actual (desired) and the predicted value. There are a number of measures of accuracy in the forecasting literature and each has advantages and limitations. Because of the limitations associated with each individual measure, it is recommended to use multiple performance measures in a particular problem [9]. The following are some of the most popular measures across the literature and the ones we used in the experiment:

- Mean absolute error (MAE) $= \frac{1}{n}\sum|e_i|$

  As the name suggests, the mean absolute error is an average magnitude of the absolute errors$e_i = |f_i - y_i|$, where f_i is the prediction and y_i the true value, without considering their direction. The MAE is a linear score which means that all the individual differences are weighted equally in the average.

- Root mean squared error (RMSE) $= \sqrt{\frac{\sum e_i^2}{n}}$

  The RMSE is more sensitive than other measures to the occasional large error: the squaring process gives disproportionate weight to very large errors.

- Mean absolute percentage error (MAPE) $= \frac{1}{n}\sum\left|\frac{e_i}{y_i}\right|(100)$

  It indicates how big is the error e_i compared to the true value y_i. Since it is expressed in generic percentage terms it is often preferred for purposes of reporting because it will make sense even to someone who has no idea what constitutes a "big" error in terms of MAE and RMSE.

- Direction Accuracy (DAC) $= \frac{1}{n}\sum a, \quad a = \begin{cases} 1 & if \;\; (y_{i+1} - y_i)(f_i - y_i) > 0 \\ 0 & otherwise \end{cases}$

  As the name suggests it measures how accurate is the forecaster in predicting the direction of movement of the stock and as the MAPE it is expressed in percentage. It is not commonly used, however, we decided to included it in or experiment because it seemed interesting and useful.

For the first three measures the smaller the number the better it is, while for the DAC it is the opposite, the larger the number the better it is.

3.7 Step 7: Neural network training

As explained in section 2.4, a neural network must be trained before being used in any forecasting problem. This process requires the model to go through many iterations of observations and adjust the weights of the arcs between the neurons so that the error function could reach a global minimum. Unless the model is over-fitted, this set of weights should provide good generalization. The back-propagation algorithm, which is the most popular training method and the one used in the experiment, uses a gradient descent method which adjusts the weights to move down the steepest slope of the error surface. However, it is not a trivial task to reach the global minimum as the algorithm can get stuck in local minima. This section discusses some of the essential parameters of the training algorithm. In particular we will illustrate when to stop training the neural network and the selection of the learning and momentum values.

3.7.1 Number of training iterations

Kaastra and Boyd point out that, essentially, there are two main ways to determine the ideal point at which training should be stopped. The first one is when the convergence point is reached, that is the point at which the error function ceases to improve after a reasonable number of starting weights have been randomly selected [54]. The second method suggests using a series of train-test interruptions [53] [71]. After a fixed number of iterations training should be stopped, the prediction accuracy of the network should then be evaluated on the testing set. After evaluation training should restart and the strategy should be repeated until an acceptable prediction accuracy is reached. However, many criticize that this additional train-test interruptions could worsen the error on the testing set since researchers can't know if additional training could improve the prediction accuracy of the network, especially since weight are chosen at random. Another disadvantage is that the researcher has to two extra parameters two consider; namely the point at which training should be stop and the strategy to evaluate which of the train-test networks achieve the best performance. On the other hand, the train-test approach requires less training time [11]. Finally, the main advantage of the convergence approach is that there is a higher chance of avoiding local minima.

In practice, however, since the convergence approach is time consuming and there is no guarantee that it will reach the global minimum, it is recommended to set a limit to the maximum number of runs of the approach. As summarized by Kaastra and Boyd [11] many studies that mention the number of training iterations (also known as epochs) reported convergence from 85 to 5000 iterations [72] [73]. However, the range is very wide as 50,000 and 191,400 iterations [70] [74] and training iterations of 60 hours have also been reported [75]. As it is evident, the results from the literature are widespread. This is because many parameters affect training and very often their values differ from study to study. Thus one can understand why it is difficult to determine a common optimal value for the maximum number of runs. Following Maciel and Ballini [76] we utilized 500, 2000, 5000 and 10000 iterations.

3.8 Step 7: Learning rate and momentum

As illustrated in section 2.5, the learning rate is a constant that helps to identify the magnitude of the adjustments to the weights. As an analogy to the back-propagation training algorithm, consider the example in Figure 4. Suppose that you want to throw a ball from point A to point C. It is worth remembering that it is not possible to represent the error surface in a graphical format because in reality it is multidimensional.

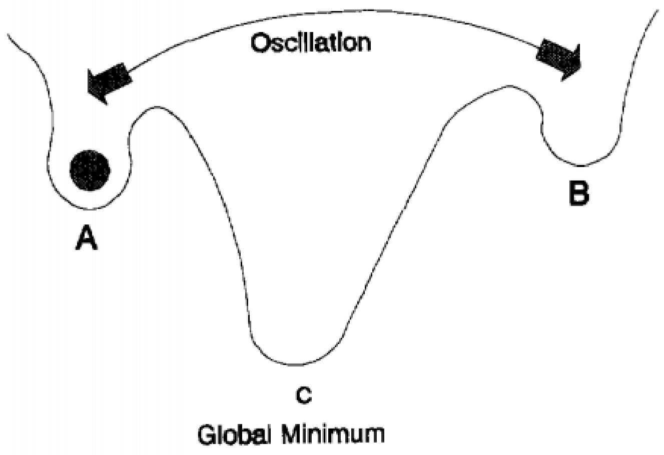

Figure 4: Simplified graphical representation of a neural network error surface [38]

The force of the ball is analogous to the learning rate. If you throw the ball with too much force this will overpass its target and can either get stuck in point B or can oscillate between points A and B. On the other hand, if the ball is thrown with too little force it won't be able to escape from point A. During training, is the error function is changing wildly without showing any continued improvement means that the learning rate is too high. Intuitively, little or no improvement is an indication that the learning rate is too small. In either case, the optimal solution would be to constantly adjust the learning rate during training or retrain the network using weights selected at random and another learning rate value [11].

It is very popular to add a momentum term to the back-propagation algorithm, this increases the learning rate value thus reducing training time but without risking to generate oscillation in the predicted error. The momentum term can be seen as the inertia generated by previous changes in the weights. It is a way of linking past to current weight changes. Large values of the momentum term will influence the adjustment in the current weight to move in the same direction as previous adjustments [38]. Moreover, it manages to suppresses side to side oscillations by filtering out high-frequency variations. Another benefit of the momentum term is that, by increasing the rate at which the weights approach the global minimum, it helps the algorithm in the initial stages of the training thus speeding up the process even more. However, researches must pay attention in the choice of the momentum term because if it is too large the weight adjustments may again overshoot the minimum, due to the cumulative influences of many previous adjustments [38].

Once again we turn to the use of metaphors to make the concept clearer. Consider Figure 5 and 6. In both figures, the weight is initialized at location I, local minima are found at locations A and C, while the optimal global minimum is located at point B. The size of the ball represents the magnitude of the momentum term. Suppose we roll the small ball in Figure 5 down the curve, it may never make it over the first hill, and remain stuck at point A. Thus, the ball will never reach the global minimum represented by point B because it gets stuck at point A, which represents a local minimum.

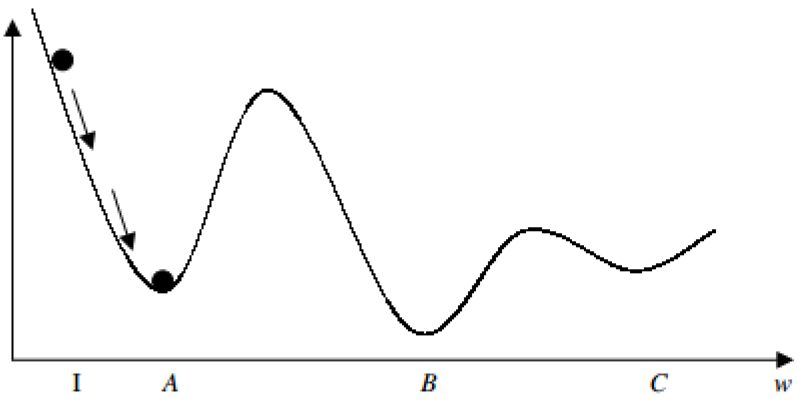

Figure 5: Small momentum may cause algorithm to undershoot global minimum [38]

Next, if we roll the large ball in Figure 6 down the curve, it may well make it over the first hill but may then have so much momentum that it overpasses the global minimum at location B and settles for the local minimum at location C.

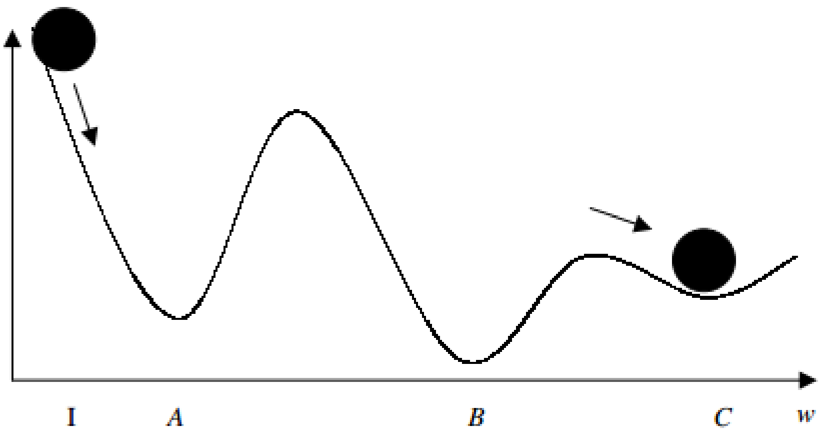

Figure 6: Large momentum may cause algorithm to overshoot global minimum [38]

These examples illustrate that careful consideration is needed in the selection of the learning rate and momentum values. Fortunately some guidelines exist. Tang et al. [32] reported that high learning rate is good for less complex data and low learning rate with high momentum should be used for more complex data series. However, because the learning rate and the momentum can assume any value between 0 and 1, it is actually impossible to do an exhaustive search to find the optimal combination of these training parameters. Therefore, common practice among researchers is to consider only selected values and experiment only with them. For example, Sharda and Patil [33] tried nine combinations of three learning rates (0.1, 0.5, 0.9) and three momentum values (0.1, 0.5, 0.9). In our experiment we tried 16 different combinations of four learning rates (0.3, 0.5, 0.7, 0.9) and four momentum values (0.2, 0.4, 0.6, 0.8).

3.9 Step 8: Implementation

Many forecasting software exists, although often they are very expensive and difficult to use. As mentioned earlier, the experiment was conducted using an open source program called WEKA, which is a collection of machine learning algorithms for data mining tasks developed by the University of Waikato [23]. The reasons for choosing it were the following:

- It is open source;
- It's interface is intuitive and easy to use;
- We wanted to assess its capabilities and understand if it could be used as an option to other more popular forecasting tools.

4 Experimental Design

The experiment consisted in using a feed-forward multilayer perceptron network trained with back-propagation for one-step ahead forecasting of financial time series. The aim was to identify, where possible, the optimal values for the free parameters of the MLP. After studying the literature and realizing that the number of free parameters is very large, we have decided to concentrate only on those parameters that we think have a major effect on the performance of the forecaster or on those for which researchers have failed to agree on a clear winner regarding its optimal value. These were:

- The sample set size;
- The number of input neurons;
- The number of hidden neurons;
- The number of training iterations;
- The value of the learning rate;
- The value of the momentum.

It is nearly impossible to optimize the value of all the parameters at the same, there would be too many combinations, therefore it will take too much time. In fact, considering that for each value we tried four values, 4096 predictions error would have been generated for each one of the fifteen time series. For this reason we decided to split the experiment into two parts: during the first part we try to optimize the first three parameters and then in the second part we optimized the other three. We thought this was a good trade off since we still generated a large number of results. Furthermore, we thought it would be a good idea to optimize closely related parameters at the same time. In fact, we optimized parameters 4, 5 and 6 together because they are all related to the network training, thus maintaining the underlying links that exist between them. A similar argument can be proposed for parameters 2 and 3 since they are related to the structure of the artificial neural network.

The experimental setup for the first part of the experiment was as follows:

- Sample set: 3, 6, 13, 25 months of daily data, which corresponded respectively to 63, 128, 275 and 527 instances.
- Input neurons (lagged variables): 2, 4, 8, 12. WEKA provides an interesting functionality called "adjust for variance" which allows the system to automatically compensate for variance in the data. It does it by taking the log of each target before creating lagged variables and building the model. It was observed that this functionality, in general, improved the accuracy of the forecaster, so we decided to use it.
- Hidden neurons: n/2, n, 2n and 3n, where n is the number of input neurons.
- Only one output neuron was used.
- The sigmoid transfer function was used for all hidden and output neurons.
- Only one hidden layer was used.
- Training set was 80% of sample set.
- Testing set was 20% of sample set.
- The "normalize numeric class" functionality was set to true: this normalized the data set to be between -1 and 1 if it was numeric, which was our case. The

normalization was only internally, thus, the output of the forecaster is presented in its original range.
- The decay functionality was set to true: this divided the starting learning rate by the iteration number, to determine what the current learning rate should be. This will cause the learning rate to decrease, thus it may stop the network from diverging from the target output, as well as improving general performance as explained in section 3.8.
- The reset functionality was set to true: this allowed the network to automatically reset itself with a lower learning rate and begin training again if it diverges from the correct answer.
- Validation threshold was set to 20: the value here indicates how many times in a row the validation set error can get worse before training is terminated. During the experiment training terminated or because either the validation threshold or the number of iterations was reached.
- Learning rate was 0.3, which was WEKA's default value.
- Momentum was 0.2, which was WEKA's default value.
- Training iterations were 500, which was WEKA's default value.

The experimental set up for the second part of the experiment was almost the same, except for a few small changes. The parameters that we tried to optimize were:

- Learning rate: 0.3, 0.5, 0.7, 0.9.
- Momentum: 0.2, 0.4, 0.6, 0.8.
- Training iterations: 500, 2000, 5000, 10000.

The values for the sample set size and for the number of the input and hidden neurons varied from one time series to the other. For each time series we used the values that gave the better performance during the first part of the experiment. We understand that this method may not be the best one, however, that is how we decided to progress due to our limited knowledge in optimization techniques. A possible improvement could have been to iterate through the experiments several times until no further significant improvement was reached for every time series. In each iteration the information from the second part of the experiment should have been used to tune the parameters for the first part of the next iteration as we did in our experiment from part one to part two. This approach would have been probably more efficient, nevertheless, due to the lack of time we couldn't implement it.

We considered the time series from 15 companies of the FTSE100. We thought that choosing the companies from the same market would make the experiment more robust. In this way we have a higher chance that changes in the performance of the forecaster are directly related to changes in the parameter since all the data belongs to the same environment. Furthermore, the data was also more consistent because the stocks of the companies were traded on the same days. The companies were the following:

- Ryanair;
- Easy Jet;
- International Consolidated Airlines Group;
- Barclays;
- HSBC;

- Lloyds;
- Betfair;
- Ladbrokes;
- William Hill;
- Tesco;
- Morrison;
- Sainsbury;
- BP;
- Royal Dutch Shell;
- BG Group;

As it can be noticed from the list below, the companies can be grouped into 5 different categories: airlines, banks, bookmakers, grocery and general merchandise retailers, oil and gas producers. It is expected that the same values of the parameters generate if not the same at least similar results for the time series of companies belonging to the same sector.

The intervals of time under consideration were:

- For the 3 months sample set: 31/01/2014 - 01/11/2013 ;
- For the 6 months sample set: 31/01/2014 - 01/08/2013 ;
- For the 13 months sample set: 31/01/2014 - 01/01/2013 ;
- For the 25 months sample set: 31/01/2014 - 01/01/2012 ;

5 Results

The performance of the forecaster was evaluated by analyzing only the results from the testing set, however, evaluation was conducted also on the training set just for the sake of curiosity. For a full set of the results please refer to the additional material. Table 1 illustrates the best results obtained from the first part of the experiment, together with the values of the parameters that generated them. In the selection of the best results, more importance was given to the MAPE because it represents a relative performance measure, therefore it can be used for comparison between different time series. Ties were solved by looking in order at the MAE, RMSE and finally at the DAC. The parameters in Table 1 refer only to those that produced the best result based only on the MAPE, which most of the time also generated the best results for the MAE and RMSE. Rarely, a set of parameters produced the best results for the four performance measures, but this was expected.

	MAPE	MAE	RMSE	DAC	Sample Set	Input Neurons	Hidden Neurons
Ryanair	1.5066	9.9002	12.5158	36	6	4	n/2 (2)
Easy Jet	2.1043	34.8402	41.4682	64	6	4	3n (12)
I. C. A. Group	1.6814	6.578	8.2807	62.963	13	4	3n (12)
Barclays	1.0091	2.8483	3.7716	68	6	8	3n (24)
HSBC	0.7013	4.7416	6.1995	52.8846	25	2	3n (6)
Lloyds	1.2783	0.9907	1.2397	42.3077	25	2	2n (4)
Betfair	1.2902	13.1748	17.8413	50	25	8	2n (16)
Ladbrokes	1.5669	2.7675	3.8478	61.5385	25	4	n (4)
William Hill	1.4404	5.4085	7.1074	57.4074	13	8	2n (16)
Tesco	0.646	2.1248	2.887	33.3333	3	8	2n (16)
Morrison	0.8237	2.0552	2.8843	41.6667	3	12	n (12)
Sainsbury	0.9685	3.6979	4.9991	47.1154	25	4	3n (12)
BP	0.5467	2.6522	3.8119	50	3	12	n (12)
Royal Dutch Shell	0.6632	15.0761	19.3291	28	6	4	3n (12)
BG Group	1.3989	16.428	31.1317	57.4074	13	4	2n (8)

Table 1: Best performance results obtained from the first part of the experiment

Overall, performance was good during the first part of the experiment, with the smallest and largest MAPE's values being respectively 0.5467 for BP and 2.1043 for International Consolidated Airlines Group. By analyzing the MAE and RMSE we understand that there is some variance in the magnitude of the errors since for all the time series the RMSE is greater than the MAE. Nevertheless, very large errors are unlikely to have occurred because the RMSE-MAE difference isn't large enough to indicate the presence of very large errors.

Regarding the values of the parameters tested some interesting thing can be highlighted. The optimal sample set size was in 33 % of the cases 25 months, in 27 % of the cases 6 months, in 20 % of the cases 13 months and finally in 20 % of the cases 3 months. Obviously no optimal sample size exists for all the series, this is in contrast with Nam and Schaefer [61] who found that the ANN forecaster performed better as the training size increased. However, a possible explanation for this could be that the data set was not detrended nor deseasonalized and this could have affected the performance of the network. Regarding, the single categories it is worth pointing out that possibly the optimal value for the airlines category lies between 6 and 13 months, while for the bookmakers category it is at least 13 months. Nothing can be said for the other categories.

The optimal number of input neurons was in 47 % of the cases 4 neurons, in 27 % of the cases 8 neurons, in 13 % of the cases 2 neurons and finally in the remaining 13 % of the cases 12 neurons. It could be said that probably a good value for the number of neurons lies between 4 and 8 since both values accounted for 74 % of the optimal range. Considering the airline categories the ANN forecaster produced the optimal results for all the companies when 4 input neurons where used.

Similarly, the optimal number of hidden neurons was in the 40 % of the cases 3n, in the 33 % of the cases 2n, in the 20 % of the cases n and in the 7 % of the cases n/2. This result may suggest that the optimal value for the number of hidden neurons could possibly be found between 2n and 3n since both values together accounted for 73 % of the cases.

As stated before, rarely a set of parameters' values produced the best results for the four performance measures and this behavior was expected. However, it is interesting to notice from Table 2 that in 13 out of 15 cases, that is 86 % of the cases, the values of the parameters that generated the best results for the MAPE did not produce the best performance for the DAC. Moreover, in most of the cases the difference between the reported DAC and the actual best DAC was significant. This means that it is difficult to obtain, with the same set of values, both precision in the prediction of the value of the closing price and precision in predicting the direction of change of the closing price. Probably this phenomenon can be understood better with an example.

Suppose that you have a stock A whose closing prices for day 6 and 7 were 83 and 86. Suppose also that the forecaster has correctly predicted all the previous closing prices, in this case the MAPE will be 0 % and the DAC 100 %. Now the forecaster predicts that for day 6 the closing price will be 89, in this case the MAPE will worsen to 0.6 % but the DAC will still 100 % because the direction of change was predicted correctly. After this, suppose that the forecaster predicts that the closing price for day 2 will be 87, in this case the MAPE will improve to 0.16 %. However, the DAC will get worse because the actual closing price rose from 83 to 86 while the forecaster predicted that it was going to decrease from 89 to 87. The DAC will now be 50 %. This is probably due to the back-propagation algorithm that updates the weight of the network in the direction that would minimize the error without considering if the direction is correct.

	DAC	Best DAC	Difference	Improvement (%)
Ryanair	36	57.4074	21.4074	59.465
Easy Jet	64	64	0	0
I. C. A. Group	62.963	66.6667	3.7037	5.882343599
Barclays	68	68	0	0
HSBC	52.8846	58.3333	5.4487	10.30299936
Lloyds	42.3077	58.3333	16.0256	37.87868402
Betfair	50	58.3333	8.3333	16.6666
Ladbrokes	61.5385	75	13.4615	21.87492383
William Hill	57.4074	59.6154	2.208	3.846194045
Tesco	33.3333	60	26.6667	80.00018
Morrison	41.6667	60	18.3333	43.9998848
Sainsbury	47.1154	58.3333	11.2179	23.80941263
BP	50	52.8846	2.8846	5.7692
Royal Dutch Shell	28	58.3333	30.3333	108.3332143
BG Group	57.4074	75	17.5926	30.64517815

Table 2: Differences in DAC values in the first part of the experiment

Let's now analyze the results from the second part of the experiment that we present in table 3. Remember that the values for the sample set size and for the number of the input and hidden neurons varied from one time series to the other. For each time series we used the values that gave the better performance during the first part of the experiment. These are shown in table 1.

	MAPE	MAE	RMSE	DAC	Iterations	Learning Rate	Momentum
Ryanair	1.3599	8.9513	11.1665	32	500	0.3	0.8
Easy Jet	1.0902	17.884	26.0073	64	2000	0.3	0.6
I. C. A. Group	1.6454	6.4166	8.1035	59.2593	10000	0.3	0.2
Barclays	0.9693	2.7322	3.6033	64	2000	0.3	0.2
HSBC	0.6864	4.6433	6.0986	50.9615	10000	0.3	0.2
Lloyds	1.1347	0.8814	1.1177	44.2308	2000	0.7	0.2
Betfair	1.2147	12.4796	16.4507	47.1154	10000	0.7	0.4
Ladbrokes	1.4432	2.539	3.5032	57.6923	5000	0.9	0.4
William Hill	1.4404	5.4085	7.1074	57.4074	500	0.3	0.2
Tesco	0.6178	2.0246	2.6638	33.3333	10000	0.3	0.2
Morrison	0.8237	2.0552	2.8843	41.6667	500	0.3	0.2

Sainsbury	0.9091	3.4716	4.6871	48.0769	10000	0.3	0.4
BP	0.5412	2.6273	3.8454	50	10000	0.3	0.2
Royal Dutch Shell	0.6312	14.3268	18.3922	32	500	0.3	0.4
BG Group	1.153	13.9325	27.2915	55.556	10000	0.7	0.4

Table 3: Best performance results obtained from the second part of the experiment

As it can be inferred from table 3, the optimal number of iterations was in the 47 % of the cases 10000, in the 27 % of the cases 500, in the 20 % of the cases 2000 and in the remaining 6 % of the cases 5000. It is evident that no optimal number of iterations can be identified based on the results. However, it is important to remember that the training of the ANN could have been stopped if the validation threshold has been reached. Therefore, there could be some cases in which the number of iterations reported in table 3 was not fully completed. Unfortunately, WEKA does not allow us to see which of the two events caused the training to stop.

Results are quite different when considering the learning rate. In fact, the optimal value for the learning rate was in the 73 % of the cases 0.3, in the 20 % of the cases was 0.7 and in the 7 % of the cases was 0.9. It is interesting to notice that 0.5 never represented the best learning rate. On the other hand, it is easy to see that 0.3 is the predominant learning rate. It was also the default value of the WEKA forecasting tool. This result is in contrast with Kaastra and Boyd [11] who said that it is common practice to start training with a higher learning rate such as 0.7 and decrease as training progresses. It is interesting to highlight that the best learning rate for the airlines and retailers category was 0.3.

Similar results were obtained for the momentum term. The optimal value was found to be 0.2 in the 53 % of the cases, 0.4 in the 33 % of the cases, 0.6 in the 7 % of the cases and 0.8 in the remaining 7 % of the cases. It is probable that smaller values represent the optimal point for the momentum term, in particular values between 0.2 and 0.4 since together both values covered 86 % of the cases.

Again in 13 out of the 15 time series the values of the parameters that generated the best results for the MAPE did not produce the best performance for the DAC. This fact further consolidates our hypothesis regarding the inability of the network to precisely predict the value of the closing price and the direction of the change with the same set of parameters' values.

	DAC	Best DAC	Difference	Improvement (%)
Ryanair	32	48	16	50
Easy Jet	64	68	4	6.25
I. C. A. Group	59.2593	62.963	3.7037	6.249989453
Barclays	64	68	4	6.25
HSBC	50.9615	52.8846	1.9231	3.773633037
Lloyds	44.2308	50	5.7692	13.04339962
Betfair	47.1154	53.8462	6.7308	14.28577493

Ladbrokes	57.6923	61.5385	3.8462	6.666747556
William Hill	57.4074	57.4074	0	0
Tesco	33.3333	41.667	8.3337	25.001125
Morrison	41.6667	58.3333	16.6666	39.999808
Sainsbury	48.0769	49.0385	0.9616	2.00012896
BP	50	50	0	0
Royal Dutch Shell	32	52	20	62.5
BG Group	55.556	59.2593	3.7033	6.665886673

Table 4: Differences in DAC values in the second part of the experiment

Finally, it is interesting to notice that at the end of the second part of the experiment there was an increase in performance for the ANN. In 13 out of 15 time series, 86 % of the cases, the values of the MAPE showed a decrease, which in some cases was even large. For example, the performance on Easy Jet improved by 48 % and on BG Group by 17 %. No improvement was observed only for William Hill and Morrison's time series. This is an interesting result since it shows that the methodology used for parameter optimization could be a viable option for other researchers. As explained earlier, it could be improved by transforming it in an iterative process until convergence is reached.

	1st MAPE	2nd MAPE	Difference	Improvement (%)
Ryanair	1.5066	1.3599	0.1467	9.737156511
Easy Jet	2.1043	1.0902	1.0141	48.19179775
I. C. A. Group	1.6814	1.6454	0.036	2.141072915
Barclays	1.0091	0.9693	0.0398	3.944108612
HSBC	0.7013	0.6864	0.0149	2.124625695
Lloyds	1.2783	1.1347	0.1436	11.23366972
Betfair	1.2902	1.2147	0.0755	5.851805922
Ladbrokes	1.5669	1.4432	0.1237	7.894568894
William Hill	1.4404	1.4404	0	0
Tesco	0.646	0.6178	0.0282	4.365325077
Morrison	0.8237	0.8237	0	0
Sainsbury	0.9685	0.9091	0.0594	6.133195663
BP	0.5467	0.5412	0.0055	1.006036217
Royal Dutch Shell	0.6632	0.6312	0.032	4.82509047
BG Group	1.3989	1.153	0.2459	17.57809708

Table 5: Performance improvement between the first and second part of the experiment

6 Conclusion

The research entailed the development of a system that could estimate the next day closing stock price performance. Fifteen different companies from the FTSE100 were considered, there were three airlines, three banks, three bookmakers, three retailers and three oil and gas companies. However, the aim of the research was to identify, if possible, optimal values for some of the free parameters of the artificial neural network. More precisely, a multilayer feed-forward artificial neural network trained with the back-propagation algorithm was used.

The development methodology utilized involved, initially, varying the size of the sample set, the number of the input neurons and the number of the hidden neurons. Thereafter, the number of training iterations, and the values of the learning rate and of the momentum were varied. Mixed results were obtained for the different parameters as shown in Table 6.

	Likelihood of Optimal Values	Possible Optimal Value
Sample Set Size (months)	25 (33%), 6 (27%), 13(20%), 3 (20%)	No Optimal Value Identified
Number of Input Neurons	4 (47%), 8 (27%), 2 (13%), 12 (13%)	4 ≤ x ≤ 8 (74%)
Number of Hidden Neurons	3n (40%), 2n (33%), n (20%), n/2 (7%)	2n ≤ x ≤ 3n (73%)
Number of training Iterations	10000 (47%), 500 (27%), 2000 (20%), 5000 (6%)	No Optimal Value Identified
Learning Rate Values	0.3 (73%), 0.7 (20%), 0.9 (7%), 0.5 (0%)	0.3 (73%)
Momentum Term Values	0.2 (53%), 0.4 (33%), 0.6 (7%), 0.8 (7%)	0.2 ≤ x ≤ 0.4 (87%)

Table 6: Experiment Results Summary

The best result was obtained for the learning rate, for which 0.3 represented the optimal value in 73 % of the cases. On the other hand, no optimal value could be identified for the sample set size and for the number of training iterations. For the rest of the parameters it was possible to narrow the range in which the optimal values could lie. We understand that parameter optimization of artificial neural networks requires lots of experimentation because the performance of the networks depend on many factors. Therefore, it is almost impossible that a particular set of parameters' values could represent the optimal choice for every time series. However, the results seem to suggest that there exist some parameters' values or range of values that represent the optimal choice for most of the models. Therefore, these results are important and useful because they can be used by other researchers as a starting point when building a neural network for financial time series forecasting, thus narrowing the search space and making the process less time consuming.

Another goal of the experiment was to investigate if there exists a set of parameters' values that produce optimal performances for time series of companies from the same sector. The results obtained suggested that these similarities exist for some of the companies. For example it was observed that the best value for the learning rate was 0.3 for all the three airlines and for all the three retailers, as well as 0.2 was the optimal value for the momentum term for all three banks. Furthermore, 4 input neurons turned out to be the best choice for all three airlines. This discovery can be exploited in order to forecast closing price movements of companies that have recently entered the market or have issued an IPO (Initial Public Offering) for which there is not enough past data or there isn't anything at all. For example, investors could train the network on data from companies of the same sector and then use the trained network to predict the development of the stock of the new company.

Finally, in order to perform the experiment we used a piece of software called WEKA, which is a collection of machine learning algorithms for data mining tasks developed by the University of Waikato. We would like to express our opinion about it. We found its forecasting plug-in very easy and intuitive to use, the interface was clear and simple and its computational power was adequate for the task. However, we realize that its capabilities were limited and we have identified some major issues:

- Detrending and deseasonalization was not present in the list of available preprocessing techniques. These two are very common preprocessing techniques in classification and regression problems, therefore it would have been very useful to have them.
- There is not extensive support available on the web, in particular regarding the forecasting plug-in. Furthermore, there is a lack of active community.
- It was not possible to use a validation set. In fact, WEKA allows the user to split the sample set only into training and testing set.
- The collection of machine learning algorithms is small, in particular for regression problems. Most of the newest methods such as genetic algorithms and support vector machine, that are gaining popularity, are not present at all.
- Customization on the algorithms is limited and is concentrated around the basic parameters. For example, the only artificial neural network found was the multilayer perceptron trained by back-propagation and which uses the sigmoid transfer functions, which is the most popular. It would have been useful to allow the user to change the type of network, the training algorithm and the activation functions too.
- While training the ANN it was not possible to understand which of the stopping criteria was reached first (validation threshold or number of iterations).
- The set of performance measures was limited.

Maybe some of the functionalities are indeed present, however, if this is the case, we weren't able to find them, therefore there is a problem with the accessibility of the application. We also understand that WEKA, being and open source project, can be expanded and every user can add his own algorithms and functionalities. However, the major reason for using WEKA is that everything is ready for the user and he doesn't have to spend time coding the algorithm. If this is the case, then it is better to use other tools such as Matlab and R, that are more powerful, have better support, a greater community, have been there for quite a while and are up to date with the latest improvements in the field. So, in our opinion, WEKA is a useful tool for beginners and for someone who wants to play with forecasting for a while, nevertheless, it is not recommended for experts and for more serious purposes.

7 Future Work

We acknowledge that our experiment, although not perfectly designed, has produced encouraging results. We believe that a similar experiment should be carried out using a more systematic and professional parameter optimization strategy. Furthermore, below there are some other suggestions that we think could be interesting:

- Repeat the experiment with detrended and deseasonalized data sets. This will not only improve the performance of the ANN, which is always important, but it could help with the identification of an optimal sample set size.
- Repeat the experiment with more time series, this will make the results more reliable. In our defense, it was difficult to find companies with similar characteristics such as magnitude and sector that also belong to the same market.
- Repeat the experiment optimizing other parameters, for example more than one hidden layer can be used, the size of the training and testing set can be changed, different transfer functions can be employed.
- Repeat the experiment optimizing more parameters simultaneously.
- The results obtained in our experiment could be analyzed in more depth in order to discover similarities in the time series of companies for which the optimal value was the same.
- Repeat the experiment on similar companies from different markets for example from the NYSE or the NASDAQ.

8 References

[1] Y. S. A.-M. a. A. F. Atiya, "Introduction to Financial Forecasting," *Applied Intelligence,* vol. 6, no. 3, pp. 205-213, July 1996.

[2] P. B. P. a. T. Marwala, "Forecasting closing price indices using neural networks," in *IEEE Conference on Systems, Man and Cybernetics*, Taipei, 2006.

[3] J. J. a. E. Kyper, "Evidence on the seasonality of stock market prices of firms traded on organized markets," *Applied Economics Letters,* vol. 12, no. 9, pp. 537-543, 2005.

[4] J. S. A. a. R. A. Pohlman, "A Note on the Price Behaviour of far Eastern Stocks," *Journal of international Business studies,* pp. 103-108, February 1978.

[5] S. Basu, "Investment Performance of common Stocks in relation to Their Price-Earning Ratios: A Test of the Efficient Market Hypothesis," *The Journal of Finance,* vol. 32, no. 3, pp. 663-682, June 1997.

[6] B. H. Solnik, "Note on the validity of the random walk for European stock prices," *The Journal of Finance,* vol. 28, no. 5, pp. 1151-1159, December 1973.

[7] M. T. S. e. al., "Testing the Weak Form of Efficient Market Hypothesis: Empirical Evidence from Asia-Pacific Markets," *International Research Journal of Finance and Economics,* no. 58, December 2010.

[8] D. M. e. al., Forecasting and Time Series Analysis, McGraw-Hill, 1990.

[9] G. Z. e. al., "Forecasting with artificial neural networks: The state of the art," *International Journal of Forecasting ,* vol. 14, no. 1, pp. 35-62, March 1998.

[10] J. J. M. M. e. al., "Artificial neural networks applied to forecasting time series," *Psicothema,* vol. 23, no. 2, pp. 322-329, 2011.

[11] I. K. a. M. Boyd, "Designing a neural network for forecasting financial and economic time series," *Neurocomputing,* vol. 10, pp. 215-236, 1996.

[12] S. M. a. G. Mani, "Financial forecasting using genetic algorithms," *Applied Artificial Intelligence,* vol. 10, no. 6, pp. 543-565, 1996.

[13] G. B. a. R. Khaled, "Stock price prediction using genetic algorithms and evolution strategies," in *International Conference on Genetic and Evolutionary Methods*, Las Vegas, 2012.

[14] W. H. a. S. Yu, "Support Vector Regression for Financial Time Series Forecasting," *International Federation for Information Processing,* vol. 207, pp. 825-830, 2006.

[15] D. N. Chorafas, Chaos theory in the financial markets: Applying fractals, fuzzy logic, genetic algorithms, Swarn simulation & the Monte Carlo method to manage markets., Probus Publishing Company, 1994.

[16] R. H. a. B. Nath, "Stock Market Forecasting Using Hidden Markov Model: A New Approach," in *International Conference on Intelligent Systems Design and Applications*, Wroclav, 2005.

[17] K. Kim, "Financial time series forecasting using support vector machines," *Neurocomputing,* vol. 55, no. 1-2, pp. 307-319, September 2003.

[18] L. C. a. F. Tay, "Financial Forecasting Using Support Vector Machines," *Neural Computing & Applications,* vol. 10, no. 2, pp. 184-192, May 2001.

[19] M. T. e. al., "Performance comparison of artificial neural network(ANN) and support vector machines (SVM) models forthe stock selection problem: An application on

theIstanbul Stock Exchange (ISE) - 30 index in Turkey," *African Journal of Business Management,* vol. 6, no. 3, pp. 1191-1198, January 2012.

[20] J. K. Mantri, "Comparison between SVM and MLP in Predicting Stock Index Trends," *International Journal of Science and Modern Engineering ,* vol. 1, no. 9, August 2013.

[21] N. K. A. e. al., "An Empirical Comparison of Machine Learning Models for Time Series Forecasting," *Econometric Reviews,* vol. 29, no. 5-6, pp. 594-621, 2010.

[22] B. K. e. al., "Financial time series forecasting with machine learning techniques: A survey," in *European symposium on artificial neural networks: Computational and machine learning*, Bruges, 2010.

[23] T. U. o. Waikato. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed 24 April 2014].

[24] K. H. e. al., "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks,* vol. 2, pp. 359-366, 1989.

[25] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks,* vol. 4, pp. 251-257, 1991.

[26] K. Hornik, "Some new results on neural network approximation," *Neural Networks,* vol. 6, pp. 1069-1072, 1993.

[27] M. J. C. H. a. H. E. Root, "An Adaptive Data Processing System for Weather Forecasting," *Journal of Applied Meteorology and Climatology,* vol. 3, pp. 513-523, 1964.

[28] D. E. R. e. al., "Learning representations by back-propagation errors," *Nature,* vol. 323, pp. 533-536, October 1986.

[29] P. J. Werbos, *Beyond regression: new tools for prediction and analysis in the behavioral sciences,* 1974.

[30] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks,* vol. 1, no. 4, pp. 339-356, 1988.

[31] A. L. a. R. Farber, "Nonlinear signal processing using neural networks: Prediction and system modelling," in *IEEE international conference on neural networks*, San Diego, 1987.

[32] Z. T. e. al., "Times series forecasting using neural networks vs. Box-Jenkins methodology," *Simulation,* vol. 57, no. 5, pp. 303-310, November 1991.

[33] R. S. a. R. B. Patil, "Connectionist approach to time series prediction: An empirical test," *Journal of Intelligent Manufacturing ,* vol. 3, pp. 317-323, 1992.

[34] Z. T. a. P. A. Fishwick, "Feedforward neural nets as models for time series forecasting," *Journal on Computing,* vol. 5, no. 4, pp. 374-385, 1993.

[35] B. K. W. a. Y. Selvi, "Neural network applications in business: A review and analysis of the literature," *Information & Management,* vol. 34, pp. 129-139, 1998.

[36] A. C. e. al., "Artificial neural network and the financial markets: A survey," *Managerial Finance,* vol. 26, pp. 32-45, 2000.

[37] T. M. Mitchell, Artificial Neural Networks, 1st Edition ed., McGraw-Hill , 1997, pp. 81-126.

[38] D. T. Larose, Discoverin Knowledge in Data: An Introduction to Data Mining, New Jersey: John Wiley & Sons Inc., 2004.

[39] R. D. R. a. R. J. Marks, 1. Neural Smithing: Supervised Learning in Feedfor - ward Artificial Neural Networks, Cambridge, MA: MIT Press, 1999.

[40] J. S. a. R. Dow, "Neural net pruning–Why and how?," in *IEEE International Conference on Neural Networks* , San Diego, 1988.

[41] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks ,* vol. 1, no. 2, pp. 239-245, 1990.

[42] A. S. W. e. al., "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems* , Denver, 1990.

[43] R. Reed, "Pruning algorithms – A survey," *IEEE Transactions on Neural Networks ,* vol. 4, no. 5, pp. 740-747, 1993.

[44] M. C. e. al., "Neural modeling for time series: A statistical stepwise method for weight elimination," *IEEE Transactions on Neural Networks,* vol. 6, no. 6, pp. 1355-1364, November 1995.

[45] A. R. e. al., "A polynomial time algorithm for the construction and training of a class of multilayer perceptrons," *Neural Networks,* vol. 6, no. 4, pp. 535-545, 1993.

[46] "A procedure for determining the topology of multilayer feedforward neural ne tworks," *Neural Networks,* vol. 7, no. 2, pp. 291-300, 1994.

[47] N. M. e. al., "Network information criterion - determining the number of hidden units for an artificial neural network model," *IEEE Transactions on Neural Networks,* vol. 5, no. 6, pp. 865-872, 1994.

[48] G. F. M. e. al., "Designing neural networks using genetic algorithms," in *3rd International conference on Genetic algorithms*, San Francisco, 1989.

[49] Z. G. a. R. Uhrig, "Using genetic algorithm to select inputs for neural networks," in *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks*, Baltimore, 1992.

[50] A. J. Jones, "Genetic algorithms and their applications to the design of neural networks," *Neural Computing and Applications ,* vol. 1, pp. 32-45, 1993.

[51] W. S. e. al., "Application of genetic algorithms to the construction of topologies for multilayer perceptron," in *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms* , 1993.

[52] B. L. M. H. a. J. M. J. Murre, "The design and evolution of modular neural network architectures," *Neural Networks,* vol. 7, pp. 985-1004, 1994.

[53] G. J. Deboeck, Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets, New York: Wiley, 1994.

[54] T. Masters, Practical Neural Network Recipes in C++, New York: Academic Press, 1993.

[55] A. Blum, Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems, New York: Wiley, 1994.

[56] M. M. N. a. W. T. Illingworth, A Practical Guide to Neural Nets, Reading, MA: Addison Wesley, 1991.

[57] "Yahoo Finance," Yahoo, [Online]. Available: https://uk.finance.yahoo.com. [Accessed 5 May 2014].

[58] R. H. a. A. Kraus, "Measuring Event Impacts in Thinly Traded Stocks," *Journal of Finance and Quantitative Analysis,* vol. 23, no. 1, 1988.

[59] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[60] G. P. Z. a. M. Qi, "Neural network forecasting for seasonal and trend time series," *European Journal of Operational Research,* vol. 160, no. 2, pp. 501-514, January 2005.

[61] K. N. a. T. Schaefer, "Forecasting international airline passanger traffic using neural networks," *Logistics and Trasportation,* vol. 31, no. 3, pp. 239-251, 1995.

[62] D. S. e. al., "A neural network short-term load forecaster," *Electric Power Systems Research,* vol. 28, pp. 227-234, 1994.

[63] X. Zhang, "Time series analysis and prediction by neural networks," *Optimization methods and software,* vol. 4, pp. 151-170, 1994.

[64] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematical Control Signals Systems,* vol. 2, pp. 303-314, 1989.

[65] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine,* vol. 4, no. 2, pp. 4-22, 1987.

[66] F. S. Wong, "Time series forecastin using backpropagation neural networks," *Neurocomputing,* vol. 2, pp. 147-159, 1991.

[67] S. Kang, *An investigation of the Use of Feedforward Neural Networks for Forecasting,* Kent State University, 1991.

[68] J. O. Katz, "Developing Neural Network Forecasters for Trading," vol. 10, no. 4, pp. 58-70, 1992.

[69] K. C. e. al., "Forecasting the behaviour of multivariate time series using neural networks," *Neural Networks,* vol. 5, pp. 961-970, 1992.

[70] C. C. Klimasauskas, "Applaying neural networks," in *Neural Networks In Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*, 1st ed., Chicago, Probus, 1993, pp. 64-65.

[71] L. Mendelsohn, "Training neural networks," *Technical Analysis of Stocks and Commodities,* vol. 11, no. 11, pp. 40-48, 1993.

[72] G. J. D. a. M. Cader, "Trading U.S. treasury notes with portfolio of neural net models," in *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, New York, Wiley, 1994, pp. 102-122.

[73] K. L. K. a. J. W. Uhrig, "Cash soybean price prediction with neural networks," in *Applied Commodity Analysis, Price Forecasting. and Market Risk Management*, Chicago, 1994.

[74] M. D. O. a. R. Sharda, "A neural network model for bankruptcy prediction," in *1992,* San Diego, IEEE International Conference on Neural Networks.

[75] L. H. e. al., "Futures trading with a neural network," in *Applied Commodity Analysis, Price Forecasting, and Market Risk Management*, Chicago, 1993.

[76] L. S. M. a. R. Ballini, *Design a neural network for time series financial forecasting: accuracy and robusteness analysis,* Sao Paulo: Instituto de Economia, Universidade Estadual de Campinas, 2008.