Training Artificial Neural Networks with Genetic Algorithms for Stock Forecasting

A comparative study between genetic algorithms and the backpropagation of errors algorithms for predicting stock prices

TIM OLSSON
KONRAD MAGNUSSON
Training Artificial Neural Networks with Genetic Algorithms for Stock Forecasting

A comparative study between genetic algorithms and the backpropagation of errors algorithm for predicting stock prices

Konrad Magnusson and Tim Olsson

Degree Project in Computer Science, DD143X
Supervisor: Pawel Herman
Examiner: Örjan Ekeberg

CSC, KTH 2016-05-09
Abstract

Accurate prediction of future stock market prices is of great importance to traders. The process can be automated using artificial neural networks. However, the conventional backward propagation of errors algorithm commonly used for training the networks suffers from the local minima problem. This study investigates whether investing more computational resources into training an artificial neural network using genetic algorithms over the conventional algorithm, to avoid the local minima problem, can result in higher prediction accuracy. The results indicate that there is no significant increase in accuracy to gain by investing resources into training with genetic algorithms, using our proposed model.
Träning Av Artificiella Neuronnät I
Syfte Att Förutspå Aktiemarknaden

Sammanfattning

Contents

Abstract i

1 Introduction 1
   1.1 Problem Statement ........................................ 2
   1.2 Scope ...................................................... 2

2 Background 3
   2.1 The stock market .......................................... 3
   2.2 Artificial neural networks (ANN) .......................... 4
   2.3 The backpropagation algorithm ............................. 5
   2.4 Evolutionary and Genetic Algorithms ....................... 6
   2.5 Related work .............................................. 7

3 Method 9
   3.1 Data ....................................................... 9
   3.2 Training ANNs using GA ................................... 10
   3.3 Network Setup ............................................ 11
   3.4 Performance Metrics ...................................... 12
      3.4.1 Mean Squared Error (MSE) ............................ 13
      3.4.2 Mean Absolute Percentage Error (MAPE) ............... 13
      3.4.3 Sum of Losses and Gains (SLG) ....................... 13
      3.4.4 T-test ................................................. 14

4 Results 15
   4.1 Models benchmarked on S&P 500 index ..................... 15
   4.2 Models benchmarked on Ericsson B stock .................. 17
   4.3 T-test ..................................................... 19
      4.3.1 S&P 500 Index ........................................ 19
      4.3.2 ERIC B .............................................. 19

5 Discussion 20
   5.1 Discussion of Results ..................................... 20
   5.2 Discussion of Method ...................................... 21
   5.3 Future research ........................................... 21

6 Conclusion 23
Bibliography 24
1. Introduction

The stock market plays an essential role in the global financial system as it allows companies to raise financial capital by trading shares of ownership in the company. However, it is not only important for companies themselves, but it is also used as an investment platform for both banks and private investors. To get a good return on investment in the stock market, accurate prediction of future prices is vital. Predicting the stock market is, however, a difficult task since many factors affect the price movements. Several approaches have been tried where a more common approach is using technical analysis[1].

Technical analysis is the means of analyzing statistics generated by market movement to predict the stock market. Since this is a mathematical process, it can be automated using computers. Today, three-quarters of the trading volume on NASDAQ is done by various forms of automated systems [2]. One approach used to predict the stock markets with technical analysis is using machine learning [3]. Among many, one family of models used are artificial neural networks (ANNs), which have shown promising results [4]. These networks aim to emulate the way a biological brain works by learning from examples. The networks are given input data and corresponding known true values during a training phase. The trained model is then used on real world data to predict future stock prices.

However, the most common way of training ANNs with the backpropagation algorithm runs the risk of not finding global minima of the error function, and instead getting stuck in local minima [5]. This might cause a problem when using ANNs for stock market prediction as stock markets are complex, hard to predict processes. Some even argue that they are not only complex, but chaotic processes [6][7] and, therefore, generate complex error surfaces when neural networks are applied [8].

One way to train ANNs that is less susceptible to the local minima problem is using genetic algorithms to evolve network hyperparameters [5]. However, this method is more computationally expensive and therefore slower. If sufficient increase in precision can be obtained, it may be worth investing the extra computational cost.
1.1 Problem Statement

The purpose of this study is to produce a model where ANN weights are evolved with genetic algorithms to predict stock prices, and compare its performance regarding computation and precision to regular backpropagation-trained ANNs. It may be worth the increased computing cost if the local minima problem can be avoided.

Thus, our problem statement is formulated as follows:

*Is it worth spending extra computing resources on training ANNs with genetic algorithms instead of backpropagation to gain better precision for stock forecasting?*

1.2 Scope

Because of limited resources, this study is restricted by certain factors. The analyzed models are tested on two datasets, S&P 500 Index and Ericsson B. For each stock, only the daily closing prices are used as input to the models. The prediction is the closing price of the next day, and the input is the previous five days. The performance of each model will be measured using three different error metrics (see section 3.4).

Genetic algorithms can be used to evolve network weights as well as the topology of the networks. This study will only evolve the weights, and keep the network topologies fixed when comparing the GA-trained network to the BP-trained.

Due to a restricted timeframe, we have decided to use an existing library that supports training ANNs using both backprop and genetic algorithms. The library in question is called Pybrain[9]. In order to build a better model of how the performance probability distributions compare to each other. As a consequence of limited time availability, every set of network parameters will only be trained and tested 20 times for each metric.
2. Background

The purpose of this section is to briefly introduce the stock market, artificial neural networks and evolutionary algorithms, and how they work.

2.1 The stock market

A stock market is a place where shares of publicly held companies are issued and traded. The primary purpose of a stock market is to serve as a platform for companies to trade shares of ownership in the company for financial capital. Shares are divided into stocks, which are then traded on stock exchanges through order books. An order book is a list where buy and sell orders for specific stocks are listed. If another trader decides to match an order in the order book, a trade executes and the stock price is updated to match the latest executed order. The price of the first trade of the day is called the opening price, and the price of the last trade of the day is called the closing price. Since stock prices continuously change based on transactions, it makes an attractive platform for investors. Being able to predict the stock prices is therefore of great interest.

Predicting the stock market is a complex task since several factors influence the price movements. The influencing factors are occasionally divided into three categories: fundamental factors, technical factors, and market sentiment [10]. Fundamental factors look at the core economics of a business. Technical factors include a mix of external conditions that might alter the supply and demand of a stock. Market sentiment investigates the overall attitude of investors. Prediction methods are usually based on investigating these price movement factors, and the most common ways of predicting stock prices are using fundamental analysis and technical analysis [10]. Fundamental analysis analyses the company by looking at the economics of the business. Technical analysis seeks to predict the price by looking at the past stock price using various statistical approaches.
2.2 Artificial neural networks (ANN)

ANNs are computational models based on biological neural networks\cite{11}. Biological neural networks, such as those in our brain, consist of billions of neurons. The neurons are connected to each other to form large networks. Each neuron receives pulses from other neurons through its inputs, processes the pulse and sends it along its output to other neurons. The biological terms for these inputs and outputs are dendrites and axons, and the connections between them are called synapses. The term synaptic strength refers to how sensitive the synapses are to incoming pulses. The pulses that travel along the axons of one neuron are received by the next neuron’s synapses. The pulses are either dampened or amplified depending on the receiving neuron’s synaptic strength. If the resulting weighted sum of inputs is larger than a certain threshold, the neuron is set to fire, sending its output down its axon. In the computational model, a neural network can be seen as a collection of neurons that are connected in a directed graph.

![A biological neuron](image1)

![Its artificial representation](image2)

There are several different types of ANNs\cite{13}. The type described so far where all neurons are connected in a directed graph without loops, and the network maps a set of inputs to a set of outputs are called Feedforward Neural Networks (FFNN). If loops are added to a FFNN, it becomes time-variant since previous input data sets will have altered the state of such loops, and affect the result of the current input. Such networks where time-variant features are introduced are called Recurrent Neural Networks (RNN).

However, this article covers a subset of FFNN, namely Multilayer Perceptrons (MLP)\cite{11}. MLPs are fully connected FFNNs with multiple layers. In this context, fully connected implies that every neuron in layer $n$ has a connection to every neuron in layer $n + 1$, as shown in figure 2.2.
2.3 The backpropagation algorithm

Backpropagation is a common method used to train ANNs. There are various optimization methods used in conjunction with backpropagation where the most common one is gradient descent. The training of an MLP consists of two phases. The first part is propagation, and includes forward propagation and backward propagation. The forward propagation feeds the input through the network and lets the network produce an output for each layer. Then, based on the output of each layer and the target value, an error is calculated for each node. A common error estimator used is the mean squared error. The error calculation phase is called backward propagation of errors. The second phase consists of two parts: calculating gradients and tweaking weights accordingly. The error gradients are computed on each input weight.

\[ \nabla E = \left( \frac{\partial E}{\partial \omega_1}, \frac{\partial E}{\partial \omega_2}, \ldots, \frac{\partial E}{\partial \omega_n} \right) \]

Weights are then incremented or decremented proportionally to the local gradient:

\[ \Delta \omega_i = -\gamma \frac{\partial E}{\partial \omega_i} \]

In this context, \( E \) is the error, \( \omega_i \) is the weight of input \( i \). \( \gamma \) defines the rate at which the weights are adjusted. This rate is usually referred to as learning rate.

Since gradient descent is a local search method, it may not always converge to global minima\([15]\). Finding the global minima may be favourable as it might result in a model that performs better. However, this is not always the case as finding the global minima could result in overfitting\([16]\). Overfitting means the model is too closely fit to the training data, and occurs when a model begins to memorize training data rather than learning to generalize from the underlying trend\([16]\).
2.4 Evolutionary and Genetic Algorithms

Evolutionary algorithms (EA) are a heuristic optimization technique where aspects of biological evolution like mutation and survival of the fittest are used to find solutions in a particular search space.

According to Darwin’s theory of evolution, there are some requirements that need to be fulfilled for evolution by natural selection to take place[17]. There needs to be a variation between organisms in the population, and these variations also need to be at least partially heritable. If these variations affect the individual’s ability to survive and reproduce, the population will gradually move toward a genome that allows for better survivability in the environment in question[17][18].

In the computational model version of this, i.e. EA, an instance of an algorithm can neither die nor reproduce, and so there is a requirement for a definition and implementation of these. Implementing mortality of algorithm instances is trivial – simple deletion suffices. However, this means the death of an organism is controlled by a supervisor and not the environment. To emulate natural death, rules are set for what makes an organism target less fit, and therefore more likely to die. This set of rules and grading system is commonly referred to as the fitness function.

As with the death of computational model instances, reproduction must also be implemented. For this, parents are chosen at random, and genetic operators are used to produce offspring. Genetic operators in this context are means by which the genome is altered. The traditionally used genetic operators are crossover and mutation[18]. Mutation is simply adding some form of distortion to the genome. Crossover is mixing segments of the parents’ genomes to form an offspring’s genome. Crossover is what distinguishes Genetic Algorithms (GA) from other forms of EA.

When applying GA to computational models, a population is initialized, e.g. randomly generated, and then evaluated. The population is then culled and survivors are chosen based on their fitness. These survivors are then crossbred to refill the population. To reduce the effect of inbreeding, mutation is used along with crossbreeding. The population is then run through the same evaluation, and the process repeats until satisfactory results are achieved.

In nature, many species follow a one year long cycle. For instance, most birds mate in spring and continue to care for their young until late in the summer when the chicks have grown to be able to take care of themselves. Though not all species, humans for instance, repeat this cycle annually, it is generally
easier to manage simulations where generations are initiated and terminated synchronously.

![Figure 2.3: A graphical representation of the GA cycle [19]](image)

More relevant for this article, using GA to train ANNs can be done in several different ways. One example is letting the set of ANN weight matrices belonging to one instance of the model represent one organism. The fitness function could for instance be the mean squared error (MSE, see section 3.4.1) of the result from a testing data set. The aim is then to use GA to breed organisms where the fitness is as low as possible. Mutation could be represented by randomly sampling matrix elements and rescaling or reassigning them. Crossover is easily accomplished by producing offspring matrices using elements from the parents’ matrices.

### 2.5 Related work

In 2015, R. Hafezi, J. Shahrabi, and E. Hadavandi proposed a Bat-Neural Network Multi-Agent System (BNNMAS) [20], as a model for stock price prediction. The authors describe it as an intelligent ANN model that combines various, carefully chosen parameters as ANN input. The architecture of BNNMAS consists of four layers. The first layer gathers statistical stock data, and qualitative data like news and expert knowledge. The data is sent to layer two which preprocesses the data, and votes on which data to keep based on the relevance of the data. The third layer uses the kept data from layer two as input to an ANN, whose purpose is to predict the stock price. The fourth layer then uses the trained ANN from layer three to test the model on real world data. The model is compared to simple backpropagation trained models and the authors conclude, based on statistical tests, that the BNNMAS outperforms all of them when it comes to predicting the actual stock prices.
In 2012, a hybrid model for stock market prediction, combining GA, evolutionary Neural Network (ENN), and ANNs was proposed by Asadi et al\cite{21}. In this study, the authors intend to use this model, a pre-processed evolutionary Levenberg-Marquardt neural network model (PELMNN) as a way to get past the local minima problem of the ordinary Levenberg-Marquardt neural network model (LMBP). In LMBP, the model prediction is highly dependent on the initial network weights, as it tends to converge to bad local minimas if they are initialized improperly. A standard Genetic Algorithm is used to evolve the initial network weights in order to overcome the initial weight dependency problem. In this study, six different stocks were used as testing datasets and the average error of the predicted values varied between 0.2-0.8\%.
3. Method

This section describes the process for acquiring data, building and training the model, as well as evaluating the trained models.

3.1 Data

To evaluate the consistency of results, the input data consists of one stock index and one stock. The used stock index is S&P 500 INDEX(SNP), and the used stock is Ericsson B(ERIC B). The data was obtained from Yahoo Finance[22]. The daily closing prices are used as input, and a total of two years are used as input data for training and testing. The amount of data used is based on a study [23] which identified that a range between one or two years of data gives the optimal accuracy for stock prediction using ANNs. An example dataset is shown in figure 3.1

![Figure 3.1: Example of a dataset (S&P 500 Index).](image)

The data is divided into two parts, 70% training, and 30% testing. The training part is used exclusively for learning the weights and the testing part to see how
well the model performs on real-world data without adjusting the weights. The last five days are used as input data to the neural networks. The neural networks then predicts the forthcoming day. This process is illustrated in figure 3.2. The red-lines indicate input to the ANN, and the blue line is the predicted output. Once a daily prediction has been done, the ANN moves forward one day.

\[ Y_{\text{norm}} = (Y_{\max} - Y_{\min}) \frac{X - X_{\min}}{X_{\max} - X_{\min}} + Y_{\min} \] (3.1)

3.2 Training ANNs using GA

In this study, GA is limited to search for solutions by only altering weight matrices, and not the network setup. Unlike BP, GA does not require for the activation function to be differentiable, because GA traverses the search space by altering variables based on a random number generator [8]. This opens up for more types
of activation functions, but for the sake of fair comparison in this study, the hyperbolic tangent function is used.

The process of training the ANNs using GA goes as follows [24]:

1. Create initial population of models (set of weight matrices)
2. Evaluate the population of models
3. Select best fit models for reproduction
4. Generate and evaluate new models
5. Replace least fit part of the population with models
6. Repeat step 3 to 5 until sufficient fitness or generation limit is reached
7. Return best set of matrices

In this study, a simple comparison of fitness is used to select candidate organisms for replacement. An alternate method, that more resembles nature, is to use the roulette wheel method [24], which calculates the probability of survival for each organism based on its fitness, and then selects survivors pseudo-randomly based on these probabilities. The simpler method is used to avoid the computational overhead the roulette wheel method requires.

### 3.3 Network Setup

There are two distinct network setups to take into consideration when comparing the learning algorithms, the network trained using the backpropagation algorithm and the network trained with the genetic algorithm. The chosen network topologies are generated by running a series of tests, to find which topologies give the best predictions for the BP-trained models. The same network topologies are then used for the GA-trained models. In this study, two hidden layers with 15 hidden nodes each is used for the S&P 500 benchmarks, and two hidden layers with four hidden nodes each for the Ericsson B benchmarks.

The optimal hyperparameters for each of the networks are generated by running a series of tests with varying values. For the BP-trained network, these parameters are the following: Network topology (i.e. node count, layer count), learning rate, and momentum. Similarly, for the GA-trained network, using the same topology as the BP-trained model: Population size, mutation probability, and mutation standard deviation. The tests are conducted in a brute-force fashion, choosing the runs resulting in the best values as our comparison networks. As the chosen software framework does not support an evolving until fit enough-functionality,
a value for training generations is chosen the same way as the other parameters. 10000 generations showed best results for reasonable execution time.

The initial weights of both network structures are randomly sampled from a Gaussian distribution with a standard deviation of 0.01. The activation function used for the networks is the hyperbolic tangent function (tanh). The tanh function is used mainly for two reasons: Firstly, it is antisymmetric, meaning it doesn’t suffer from a systematic bias for neurons located beyond the first layer of the network. Secondly, if the network connectivity is large, it has been shown that back-propagation learning with antisymmetric activation functions can converge faster than non-symmetric ones [25]. For both networks, MSE is used as the fitness function.

Example network setups can be seen in the following tables.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>20</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Mutation standard deviation</td>
<td>0.25</td>
</tr>
<tr>
<td>Nodes, Input layer</td>
<td>5</td>
</tr>
<tr>
<td>Hidden Layers</td>
<td>2</td>
</tr>
<tr>
<td>Hidden Nodes, Layer 1</td>
<td>4</td>
</tr>
<tr>
<td>Hidden Nodes, Layer 2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.1: Example parameters for a GA-trained network

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.01</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.1</td>
</tr>
<tr>
<td>Nodes, Input layer</td>
<td>5</td>
</tr>
<tr>
<td>Hidden Layers</td>
<td>2</td>
</tr>
<tr>
<td>Hidden Nodes, Layer 1</td>
<td>4</td>
</tr>
<tr>
<td>Hidden Nodes, Layer 2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.2: Example parameters for a BP-trained network

### 3.4 Performance Metrics

The performance of each model is measured by calculating the differences between the predicted price and the real price using various error metrics. The performance measurement of each stock is calculated individually by calculating the average of 20 runs to reduce randomness. The standard deviation of each error metric is calculated, and a T-test is conducted to draw a statistical conclusion based on the predictions.
This study is focused on answering the question whether it is worth investing extra computational effort in training with GA vs BP, and so, only the training time of each model is measured. This is done by comparing timestamps before and after relevant code being executed. As with the other performance metrics, an average of 20 runs is used for comparison between the training algorithms.

The metrics used in this study are MSE, MAPE, and SLG - described in the following sections. In the metric descriptions, the following symbols will be used:

- $Y_t$ — Actual price at time $t$.
- $\hat{Y}_t$ — Predicted price at time $t$.
- $N$ — Total sample count.

### 3.4.1 Mean Squared Error (MSE)

Mean Squared Error is one of the more commonly used error metrics. It is defined as the average of all the errors squared, as described by 3.2:

$$MSE = \frac{1}{N} \sum_{t=1}^{N} (|Y_t - \hat{Y}_t|)^2$$  \hspace{1cm} (3.2)

### 3.4.2 Mean Absolute Percentage Error (MAPE)

MAPE is related to MSE in that it calculates an average of errors. However, the main difference is that it gives the result as a percentage. The metric is defined by 3.3:

$$MAPE = \frac{100}{N} \sum_{t=1}^{N} \left| \frac{Y_t - \hat{Y}_t}{Y_t} \right|$$  \hspace{1cm} (3.3)

### 3.4.3 Sum of Losses and Gains (SLG)

SLG was proposed by Amorim Neto[26] to estimate the trend of the prediction model relative to the trend of the real values. It is defined as the mean of the losses and earnings of the model. High values indicate that the prediction is following the short term trend well. SLG is defined by 3.4:

$$SLG = \frac{\sum_{t=1}^{N} L_t}{N}$$  \hspace{1cm} (3.4)
Where

\[ L_t = \begin{cases} 
+|Y_t - Y_{t-1}|, & \text{if } (Y_t - Y_{t-1})(\hat{Y}_t - \hat{Y}_{t-1}) > 0 \\
-|Y_t - Y_{t-1}|, & \text{otherwise}
\end{cases} \]

### 3.4.4 T-test

An independent two sample t-test is conducted in order to draw a statistical conclusion whether one model more accurately predicts the correct stock prices than the other. For this study, the null hypothesis states that the mean of each models error measures are no different from each other. The null hypothesis is rejected if the predicted value is below the significance level 0.05, indicating there is a significant difference.
4. Results

In this section, the results of the study are presented. The results are presented in sets of two graphs per run: one depicts the error in dollars, the other superimposes a graph of predicted prices onto a graph of the true prices for comparison. We also present tables showing the different error measurement results, and standard deviations for each of the training algorithms. The results are shown with three significant figures. By the end of this section, the result from the t-test is presented.

4.1 Models benchmarked on S&P 500 index

A sample run using the BP-trained model is shown in Figure 4.1. The raw error in USD ($) is shown in the upper graph. The lower graph shows the real price plotted in blue, and the predicted price plotted in red. Figure 4.2 shows an example run with the GA-trained model.
Results

Figure 4.1: Example run of BP with 5 inputs, 2 hidden layers with 15 hidden neurons each, trained for 500 days, and a learning rate of 0.025.

Figure 4.2: Model trained with GA. The model has 5 inputs, and two hidden layers with 15 neurons in each. Mutation probability and standard deviation are both 0.1.

The training algorithms were tested on the same network topology, and each model is evaluated on the testing set 20 times. Table 4.1 shows the average performance measurements of these runs.
Table 4.1: Summary of performance measurements from 20 runs. In this table, \(\mu\) represents the average value of all runs, \(\sigma\) represents the observed standard deviation.

The average execution time for the BP-trained model was 3008s, and for the GA-trained model, 12550s.

### 4.2 Models benchmarked on Ericsson B stock

In this section, we present results similarly to how they are presented in section 4.1. The stock prices are however presented in the local currency SEK (Swedish Krona). Figure 4.3 shows a sample run of the BP-trained model. A sample run of the GA-trained model can be seen in Figure 4.4. The performance measurement results are presented in table 4.2.

![Error in SEK](Image)

**Figure 4.3:** Example run of BP-trained model on ERIC B stock

Both models in Figure 4.3 and Figure 4.4 used the same network topology: 5 input neurons, and two layers with 4 hidden neurons in each. The GA-trained model used a mutation standard deviation of 0.1, and a mutation probability of 0.1. The BP-trained counterpart had a learning rate of 0.025. As with S&P,
the models were trained for 500 days, and tested during a 220-day period. The average execution time for the BP-trained model was 497s, and for the GA-trained model, 2305s.

![Graph](image)

**Figure 4.4:** Example run of GA-trained model on ERIC B stock

**Table 4.2:** Summary of performance measurements from 20 runs. In this table, $\mu$ represents the average value of all runs, $\sigma$ represents the observed standard deviation.
4.3 T-test

4.3.1 S&P 500 Index

The result of the t-test on the error measurements of S&P 500 Index are presented in table 4.3. The null hypothesis can be rejected for MSE and MAPE, but not for SLG. This suggests that the BP-trained model performs significantly better than the GA-trained model when looking at the MSE and MAPE metrics.

<table>
<thead>
<tr>
<th>Error measure</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>MAPE</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>SLG</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table 4.3: T-test on the error measurement results of S&P 500 Index

4.3.2 ERIC B

The t-test results of ERIC B are presented in table 4.4. The null hypothesis can be rejected for MSE and MAPE. This implies that the BP-trained model outperforms the GA-trained model when looking at both of these error metrics. As with S&P 500 Index, the null hypothesis cannot be rejected for SLG.

<table>
<thead>
<tr>
<th>Error measure</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>MAPE</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>SLG</td>
<td>0.161</td>
</tr>
</tbody>
</table>

Table 4.4: T-test on the error measurement results of ERIC B
5. Discussion

5.1 Discussion of Results

Analyzing the results from the runs on the S&P 500 Index, the t-test shows that there is a statistically significant difference between the prediction MSE of each model. The t-test also shows that there is a statistically significant difference between the MAPE measurements from the two models. A conclusion can therefore be drawn that, for S&P 500, no precision in predicting future prices can be gained using the model proposed in this report.

Furthermore, the t-test showed that there is no statistically significant difference between the SLG measurements. If examined more closely, the results show an interesting pattern regarding SLG. The BP-trained models vary in performance, though very little, as shown by a very small standard deviation. While the BP-trained models’ results vary in SLG, the GA-trained models all produced the same result. At first, we suspected there was an error in the measurement process. After closer inspection of the GA-models’ performance, it was clear that they all showed identical behavior regarding short-term trend prediction, which explains a standard deviation of zero in SLG measurements.

Moving on to the results from ERIC B, patterns similar to those found in the results from S&P 500 can be seen. The t-test shows that there is a statistically significant difference between the two models’ performance regarding MSE. The difference in mean MSE is much larger than in the results from S&P 500. Our suspicion is that the GA-trained model is less capable of accurately predicting the market crash around day 110. This behavior can be seen in the sample runs shown in Figure 4.3 and Figure 4.4. The t-test also showed no statistically significant difference between the SLG measurements. However, on ERIC B, the behavior regarding consistency in short-term trend prediction is now shown by the BP-trained model. The GA-trained models’ performance did show a higher mean value, but in combination with the larger standard deviation, a conclusion regarding the comparative performance cannot be drawn without a more in-depth study.
5.2 Discussion of Method

This study is limited in the sense that it only takes the closing price into account. Arguably, it might not be the best idea to limit the training data to the closing price, but it should be a sufficient dataset to compare the training algorithms. Combining the closing prices with other data, might affect the results as the error surfaces would be different.

Because of limited resources and time, the amount of input data used for the experiments was constant. The amount of input data was also based on a single study. This might cause one algorithm to be favoured in case it works better on different amounts of data.

The models were all implemented in Python using the library Pybrain, and benchmarked on identical hardware. The reason we chose to work with Pybrain is that the library had implementations for both BP and GA. Using Python greatly limits us to the performance of the interpreter. There may be optimizations made behind the scenes that could favor one of the algorithms, and perhaps even harm the performance of the other. Furthermore, there could be more performance to gain from using a library that supports using a GPU to process the matrices while training with BP. However, allowing one algorithm to run on hardware that favors it would violate our constraint to using the same hardware for both implementations.

In this study, the network topologies, i.e. layer count and amount of neurons in each layer, are chosen to favor the performance of BP. There could exist other topologies that favor GA, which in turn may generate better results than what is presented here.

5.3 Future research

As this study was limited in various resources, mainly time, there are several ways to conduct a more thorough study. A large restriction in this study was the amount of iterations used, in order to draw a more decisive conclusion this amount could be increased. Broader ranges of the various hyperparameters could be tested as well. It may be worth investigating if different results can be found by using different types of ANNs.

One interesting point from the results indicate that the GA-trained models might be better at finding the short term market trends, as the SLG values where higher. There is no huge statistical difference, but it may be interesting to investigate further.
This study also limited itself to only using the last five days to predict the forthcoming day. There are plenty of different variations one could use as input data. Increasing or decreasing the amount of input days used could be one variation, trying to predict some other day besides the forthcoming day, another. Using weekly or monthly stock data could also be interesting to use as input to complement the daily data. Also, instead of using input data from consecutive days, the samples could be spaced out. The samples could for instance be spaced out linearly, but also logarithmically.

The input data was limited in this study as it only consisted of the closing prices. Including other stock data, such as the opening prices, or the high/low prices of the day might alter the results as the error surfaces would change. Other input data to test could be data generated from technical indicators. The traded volume could also be included.
6. Conclusion

Using the proposed method of training an ANN using GA proposed in this study, the precision for prediction of future prices is not higher than that of an ANN trained with BP. Furthermore, the time taken to train the formerly mentioned model is in some cases an order of magnitude greater. Thus, the results indicate that there is nothing to gain in training ANNs with GA over regular BP for daily stock price predictions. However, because of various limitations in this study, further investigation is needed in order to draw a decisive conclusion.
Bibliography


