Feature Selection for Predicting Daily Stock Market Movement

Darren Lawton

The Australian National University

Canberra, Australia

Abstract

Artificial neural networks are universal function approximators which excel at classification and regression methods. For many years neural networks have been used throughout the financial industry and academia for modelling time series data. In this paper we design and implement an neural network to predict the daily directional movement of the S&P/ASX 200. We compare two techniques for selecting an optimal set of input features to include in the network. The first technique measure neuron relevance by activity, and second technique optimises via a genetic algorithm. Both techniques are shown to reduce the network size and improve predictive performance, however the accuracy of the network generated by the genetic algorithm is far superior. Overall, given a limited dataset, our final results do not compare with similar studies.

Keywords: Neural networks, Financial forecasting, Network size, Relevance, Genetic Algorithms

1. Introduction

The efficient-market hypothesis (EMH) states that inefficiencies in asset prices are immediately exploited to zero by market participants. Therefore asset prices completely reflect all available information and any analysis of historical price data cannot be used to accurately predict future prices [1].

In reality, however, financial and economic forecasting has been explored by both academics and industry for many decades. The underlying premise of forecasting is to find functions that map historical inputs onto outputs that represent future outcomes. This implicitly assumes there is some form of persistence in future prices [2].

Artificial Neural Networks (ANNs) are universal function approximators that can map any nonlinear function [3]. Accordingly, ANNs are used for various classification and regression methods. One specific application of ANNs is financial forecasting, specifically in the predicting the movements for stock market indexes [4]. Financial time series are non-stationary, highly non-linear and noisy. Accordingly, traditional forecasting methods are not acceptable in many cases. ANNs capture non-linear relations and generalise input patterns therefore present a potential solution [5].

In this paper we design and implement an ANN to predict the daily directional movement of the S&P/ASX 200 (AXJO) index, where the AXJO index is a market-capitalisation weighted stock market index of companies listed on the Australian Stock Exchange (ASX). Designing an ANN is a highly subjective process, and involves much trial and error. This is particularly true in selecting the data requirements and hyperparameters of the network. A fundamental aspect of this paper is to compare two techniques to select an optimal, yet simplified, input vector of our model. The first technique, as introduced by Mozer and Smolensky, removes neurons from the network by measuring relevance [6]. Whilst the second technique relies on a genetic algorithm and fitness function. Thawornwong and Enke show that the performance of an ANN will actually decline when irrelevant input features are included in a model [7]. Therefore, our objective is to ensure that the input features included our model are relevant in order to obtain good predictive results.

This paper is organised as follows. Section two describes the trained ANN and each of its components. We then introduce the two techniques used to reduce and select the input features of network. Section four provides the experimental results of our network with a comparison aforementioned the aforementioned reduction techniques. Finally, section five presents our conclusion and suggestions for further work.

2. The ANN Model

The ANN model used in this paper is a three-layer multilayer perceptron (MLP) network consisting of an input layer, a hidden layer and an output layer. Such a network is capable of approximating any function to a degree of required accuracy [8]. Consistent with traditional approaches, our network applies the back-propagation (BP) algorithm to the train the ANN. BP allows us to calculate the derivative of the loss function with respect to the weights of the network, and then apply these gradients throughout the network, so as to converge the output with target values. The inherent limitations of BP are that convergence can be slow and we may be unable to escape local optima [9]. For the purpose of this paper, we accept these theoretical limitations. In the following subsections, we highlight and describe the key parameters of our network.

2.1. Data Description

In this paper, we aim to predict the daily directional movement of the ASX200. The dataset used for our model is a combination of technical indicators calculated from the daily price movement and volume statistics for the ASX200. To highlight fundamental relationships with other markets, we have also included in our dataset technical indicators from the Dow Jones Industrial Average which is a key equity index in the United States. The main technical indicators that we use for each market are moving averages for one to five days.

Prior to revealing our inputs to the model, preprocessing of the underlying data was required. The key considerations during this stage were:

- Log returns were used to scale, detrend and normalise daily price changes. Logarithmic transformations also convert multiplicative relationships into additive relationships which are believed to simplify network training [10].
- The model uses lagged values of the dependent variable as a result of autocorrelation.
- Non-market days were ignored, as lagged indicators still hold for a given trading day.

The input dataset is a curated set of technical indicators based on adapted work previously completed by Qui and Song [11]. A key criterion for the selection of indicators used were reviews of prior research studies.

The target output for the dataset is a binary value based on whether the log return for a given trading day was negative or positive:

$$output(t) = \begin{cases} 1 & \text{if } \ln(price_t/price_{t-1}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

In building our model we had 775 trading days of preprocessed data, from January 2014 to February 2018. 70% of the dataset was used for training our model. The remaining dataset was used for evaluation of the model, with 10% separated as out of sample to validate the final model.

2.2. Model Construction

The ANN that we are building is to be used for binary classification. Specifically, by providing an input vector for any given trading day, the model will predict an upward price movement (1) or, a downward price movement (0). Our model contains three layers, which should be sufficient in approximating the required function.

In terms of designing each layer, there is an infinite number of configurations for our ANN. This issue is compounded by the fact that there are no set standards for selecting a configuration based on the required use of the network.

The number of input neurons is contingent on the number and type of independent variables selected in the mode. Similarly, the number of output neurons is based on the desired number of output classes - so in our case, only one output neuron is required. That is, if the output value rounds to one then the market is predicted to increase. Conversely is if the output value rounds to zero then the market is predicted to decrease.

Determining the number of neurons in the hidden layer is a very important decision, but equally challenging. We need to balance convergence and generalisation. If we have too many neurons in our hidden layer the network will learn our training dataset, with no generalisation. Generally, the solution to determining the optimal number of hidden layer neuron is via trial and error. For our purpose, and given the constraints of this project, we simply apply a rule of thumb by using:

$$hiddenneurons = \sqrt{inputneurons \times outputneurons} \times 2$$

Activation functions determine the output, or activity, of a processing neuron. Given the nature of the underlying dataset, we require an activation function that adds non-linearity to the model. As such, we use the sigmoid activation function given that it conveniently squashes neuron output to a value between zero and one - which is in line with the target values of our dataset.

2.3. Model Training

By training our model, we want to find the set of weights between neurons that minimise the mean square error (MSE) of the model. Training via the BP algorithm is affected by many parameters such as the number of epochs, learning rates and momentum rates. Whilst there are no set standards for selecting these parameters, for the purpose of this study, we have used the following hyperparameters in order to derive our results:

Parameter	***Value***
Epochs	10000
Learning Rate	0.05
Momentum Rate	0.01
Loss Function	Mean Square Error

Table 1: ANN Selected Hyperparameters

3. Input Pruning

Despite having a curated set of preprocessed independent variables to develop our ANN, there is no guarantee of optimality. Since we have 18 features in our original input vector, there are 2^{18} feature combinations. Therefore we need to ensure that the combination of features used in our final model are relevant in order to obtain good predictive results. Thawornwong and Enke show that the performance of an ANN for financial time series prediction declines when irrelevant features are included in the model [7]. Furthermore, the more complex an ANN is, the longer it will take to train and over-fitting may potentially also occur. Accordingly, it is important to design an ANN that has reduced complexity and includes only the most relevant inputs [5]. In this vein, we introduce the following two techniques for input vector selection:

3.1. Neuron Relevance

The underlying premise of this reduction technique is to remove inputs deemed irrelevant by some measure. The measure we use is adapted from the work of Mozer and Smolensky [6], whereby the measure of neuron relevance (r_i) is based on calculating how well a network performs with a neuron compared to without the same neuron, where performance is measured by a loss function. We respectively define neuron activity and output below, where for each neuron i we attach a coefficient α_i which represents the attentional strength of a neuron's output:

$$a_j = \varphi(\Sigma(w_{ji}o_i))$$

$$o_j = \alpha_j a_j$$

We can now toggle the attentional strength of the neuron to turn it on or off, in order to calculate relevance:

$$r_i = E(\alpha_i = 0) - E(\alpha_i = 1)$$

Given the computational cost of calculating this relevance measure for every node in the network, it is reduced to the following estimation:

$$\hat{r}_i = -\frac{\delta E}{\delta \alpha_i} \text{ where } \forall \text{ i} \in \text{network}, \alpha_i = 1$$

It then follows, rather than calculating the integral of $-\delta E/\delta\alpha_i$ between $\alpha_i = 0$ and $\alpha_i = 1$, we simply evaluate the derivative at $\alpha_i = 1$ to compute $\hat{r}_i$. This estimation preserves the relation $r_i > r_j$ where neuron i is deemed to be of higher relevance to the model than neuron j.

Given α_i is fixed, we can calculate estimated neuron relevance using BP. However, we need to use a linear loss function because if we were to use MSE, the gradient will converge on zero as error decrease and consequently $\hat{r}_i$ will decrease as the error decreases. [6]

As such, we use the following linear loss function in computing relevance:

$$E = \Sigma_p \Sigma_j (t_{pj} - o_{pj})$$

Finally we rank all input neurons by their respective $\hat{r}_i$ values, the lowest ranked neuron is then deemed to be least relevant to the model.

3.2. Genetic Algorithms

Genetic algorithms (GA) are used to stochastically search for optimal solutions to a given problem. The central concept of a GA borrows from biological evolution, where aspects of a solution are encoded into a chromosome which is then mapped to a performance measure. The aim of the algorithm is to propagate the best performing chromosomes into ensuing generations until a termination condition is reached.

In this paper we encode a set of input features into a chromosome in order to train an ANN, and measure performance. Essentially we want to find the optimal subset of input features for our model via a stochastic search guided by evolutionary concepts. We note that fitness is measured by testing accuracy, therefore there is no guarantee that the final input vector will have necessarily reduced in size, since the best performing model might in fact require all input features. The key components of our GA are as follows:

- **Genetic representation**: We represent a chromosome as a binary array of length 18. Each gene represents an input feature, so that if an allele is set to one, then the feature is included in the model.
- **Fitness function**: We use the testing accuracy of the trained model to measure the fitness of a chromosome.
- **Selection**: In propagating a next generation, we bias selection towards the most-fit chromosomes; a form of elitism. This is achieved by skewing the probability that such chromosomes are selected to be included in the following generation.
- **Reproduction**: We use a crossover rate of 0.9 and mutation rate of 0.5. Our crossover method ensures that at least one parent is from the top quartile of fitness for the given generation.

Figure 1 shows a high-level structure of the GA we used in building our ANN. The algorithm produces 25 generations, each with a population of 75. In total, 1875 networks are trained and tested. Whilst this is computationally expensive, it is still significantly more efficient than searching the entire input space for optimality via brute force.

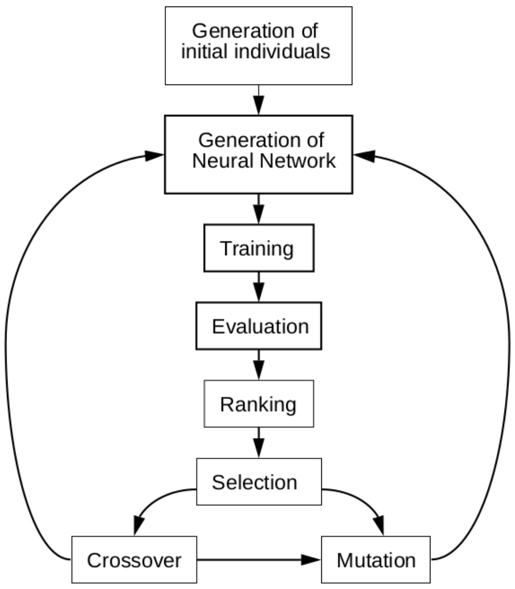
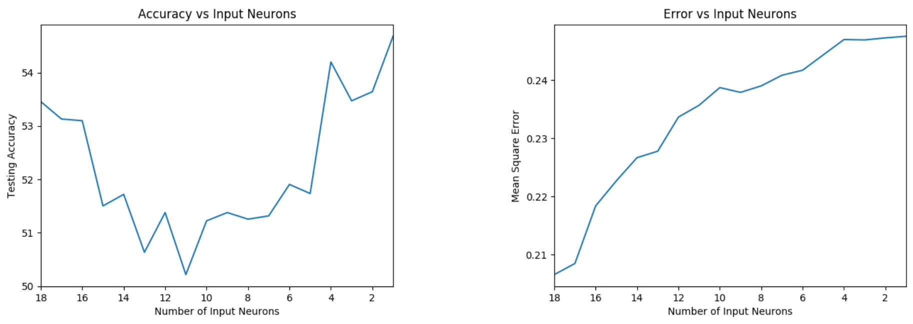

Generation of initial individuals
Generation of Neural Network
Training
Evaluation
Ranking
Selection
Crossover
Mutation

Figure 1: Structure of GA used to build ANN [12]

4. Experimental Results and Comparison

In our first experiment, we constructed an ANN using input neuron relevance as a pruning mechanism. We began with a complete input vector of 18 independent variables. Then after training and testing each network with sampling, we reduced the input vector by removing the least relevant input neuron - that is the independent variable with the lowest $\hat{r}_i$. To avoid potential overfitting from stagnant neurons in the hidden layer, we calculated the number of neurons in this layer dynamically based on formula noted in 2.2. Therefore as the number of input neurons reduced, so to did the number of neurons in the hidden layer. By iteration, we continued this process until there were no remaining input neurons. At this point, the model which achieved the highest average accuracy on the test data was returned as the final model.

In the experiment, a network containing one input neuron is returned as the final ANN. This single input feature was the lagged daily volume of the ASX200. The model achieved an average test accuracy of 54.68%, and predicted 55.13% of price movements accurately in the out of sample validation dataset. Per figure 2a, we see that testing accuracy declines after the first few input neurons are removed, but then begins to improve at a greater rate after a critical value of input neurons are removed. Concurrently, the MSE of the network steadily increases as input neurons are removed. These results agree with our hypothesis that by removing irrelevant input features, we can improve the generalisation of our model, and hence predictive performance. Furthermore, we also see a significant improvement in training times for our smaller model.



(a) Model accuracy

(b) Model error

Figure 2: Experiment one: ANN using relevance to prune input neurons

In our second experiment, we constructed an ANN using a genetic algorithm to determine the optimal subset of input features to use in our final model. Where the final model was taken to be the network with the highest fitness value in the last generation. In the experiment, a network containing 11 input features was returned as the final ANN. The network included input features from all three markets and achieved an average test accuracy of 62.02%. The predictive accuracy of the model, using the out of sample validation dataset, was 64.03%. Despite these good results, we can see in figure 3, that the max and mean fitness measures of each generation remained somewhat flat. However, given our elitist reproduction mechanisms, we expected that the fitness measure would consistently increase throughout the generations.

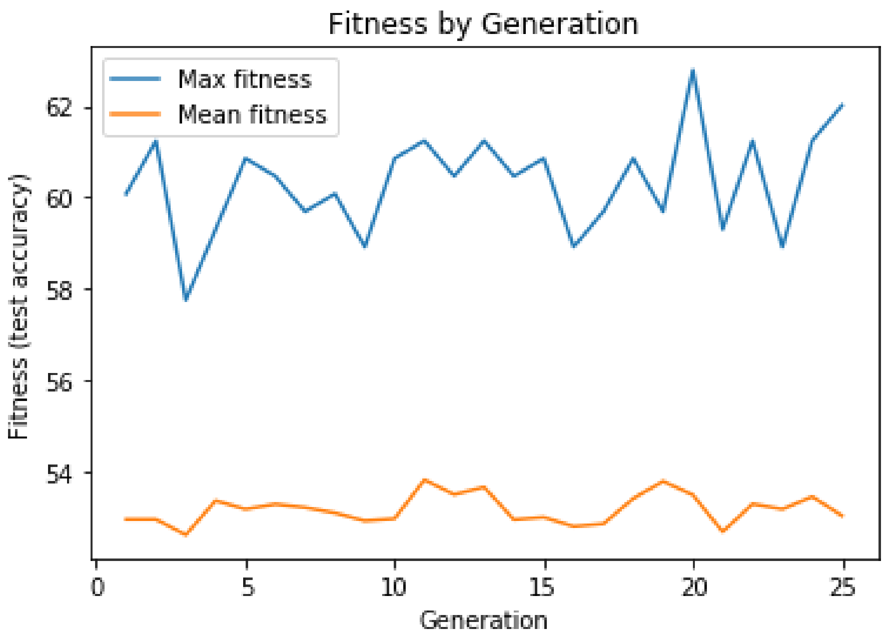


Figure 3: Average and max fitness for each generation

To determine a baseline, we trained an ANN using all 18 input features and achieved a test accuracy of 53.45%. We observe that both experiments produced improved predictive results and required less training time. The test accuracy of the network generated using input neuron relevance ($\hat{r}_i$.), was marginally better than our baseline, at 55.13%. Conversely, the test accuracy of the network generated using a GA, was significantly better than our baseline, at 62.02%.

Of interest, we note, that the single input feature included in our final model for experiment one, is also included in the final set of input features as determined in experiment two. In an indirect way, this validates $\hat{r}_i$ as a measure of neuron relevance, but also indicates that the difference in performance between both experiments may be due to how different combinations of input features interact with each other.

Despite the stronger theoretical underpinning of the $\hat{r}_i$ calculation, our genetic algorithm produces an ANN with much better predictive results. That said, when comparing our GA generated model against similar studies, the prediction accuracy underperformed, as can be seen in table 2. There is no underlying consistency in any of input data, output data, and model structure across each of these studies. Therefore identifying an exact cause of the performance discrepancy is difficult, but we suspect it may relate to the limited size and representation of the data we have used in this paper.

Studies	**Method**	**Stock Market**	**Accuracy (%)**
Huang et al. [13]	SVM	Japan	75
Kara et al. [14]	ANN	Istanbul	75.74
Qui et al. [11]	GA-ANN Hybrid	Japan	81.27
Experiment 1	ANN and relevance	Australia	55.13
Experiment 2	ANN and GA	Australia	64.03

Table 2: Comparison of results with similar studies

5. Conclusion

In this study, our aim was to construct an ANN to predict the daily directional movement of the ASX200. A key objective of this construction was to simplify the input features of the model so as improve the relevance of the included inputs, and hence prediction performance. Other benefits of this approach included efficient computation given a smaller network size and potentially enhanced generalisation. We implemented two techniques to automatically determine the optimal input features to include in our model. The first technique was neuron relevance as proposed by Mozer and Smolensky [6], and the second was a GA. Both techniques improved prediction accuracy and reduced the network size, however, the accuracy of the ANN generated by the GA was far superior. Our final results were still below expected based on similar studies.

We believe there is the opportunity to improve the model, and by extension its performance. The first opportunity would be to extend our GA to also tune the other hyperparameters of the model, such as the number of neurons in the hidden later. This would potentially have a significant impact on accuracy. In our current GA implementation, we don't see a consistent generational improvement as we would expect given our elitist approach. Accordingly, by extending the chromosome to include aspects of other hyperparameters it might expand the search space to allow for more a evident optimisation.

Secondly, since we have applied a naive approach to determining the number of neurons in the hidden layer of our network there is an obvious opportunity to apply a relevance based reduction to the neurons in this layer. This will ensure a sufficient amount of neurons for learning, but also prevent overfitting from having too many neurons.

Finally, our current implementation might just be limited by the availability of data. Therefore, there is an opportunity to grow the dataset and also review the original set of input features we have determined. However, if the dataset were to grow excessively large this may have significant impact on the time requirement for our GA based approach.

[1] B. G. Malkiel, The efficient market hypothesis and its critics, Journal of Economic Perspectives 17 (2003) 59–82.

[2] J. Yao, C. L. Tan, Guidelines for financial forecasting with neural networks.

[3] H. White, Learning in artificial neural networks: A statistical perspective, Neural Computation 1 (1989) 425–464.

[4] R. Trippi, D. Desieno, Trading equity index futures with a neural network 19 (1992) 27–33.

[5] S. T. A. Niaki, S. Hoseinzade, Forecasting s&p 500 index using artificial neural networks and design of experiments, Journal of Industrial Engineering International 9 (2013) 1.

[6] M. C. Mozer, P. Smolensky, Using relevance to reduce network size automatically, Connection Science 1 (1989) 3–16.

[7] S. Thawornwong, D. Enke, The adaptive selection of financial and economic variables for use with artificial neural networks, Neurocomputing 56 (2004) 205 – 232.

[8] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (1989) 359–366.

[9] Y. Lee, S.-H. Oh, M. W. Kim, The effect of initial weights on premature saturation in back-propagation learning, in: Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on, volume 1, IEEE, pp. 765–770.

[10] T. Masters, Practical Neural Network Recipes in C++, Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[11] M. Qiu, Y. Song, Predicting the direction of stock market index movement using an optimized artificial neural network model 11 (2016) e0155133.

[12] P. Koehn, Combining Genetic Algorithms and Neural Networks: The Encoding Problem, Master's thesis, The University of Tennessee, 1994.

[13] W. Huang, Y. Nakamori, S.-Y. Wang, Forecasting stock market movement direction with support vector machine, Computers & Operations Research 32 (2005) 2513 – 2522. Applications of Neural Networks.

[14] Y. Kara, M. A. Boyacioglu, mer Kaan Baykan, Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange, Expert Systems with Applications 38 (2011) 5311 – 5319.