01 Jan 1996

Stock Market Prediction Using Different Neural Network Classification Architectures

Karsten Schierholt

Cihan H. Dagli
Missouri University of Science and Technology, dagli@mst.edu

Recommended Citation

K. Schierholt and C. H. Dagli, "Stock Market Prediction Using Different Neural Network Classification Architectures," *Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering, 1996*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1996.
The definitive version is available at https://doi.org/10.1109/CIFER.1996.501826

Stock Market Prediction Using Different Neural Network Classification Architectures

Karsten Schierholt, Cihan H. Dagli
karsten@umr.edu, dagli@umr.edu
Smart Engineering Systems Laboratory
Engineering Management Department
University of Missouri - Rolla
Rolla, MO 65409-0370
Tel.: (573) 341-4374, Fax: (573) 341-6567

Abstract:
In recent years, many attempts have been made to predict the behavior of bonds, currencies, stocks, or stock markets. In this paper, the Standard&Poors 500 Index is modeled using different neural network classification architectures. Most previous experiments used multilayer perceptrons for stock market forecasting. In this paper, a multilayer perceptron architecture and a probabilistic neural network are used to predict the incline, decline, or steadiness of the index. The results of trading with the advice given by the network is then compared with the maximum possible performance and the performance of the index. Results show that both networks can be trained to perform better than the index, with the probabilistic neural network performing slightly better than the multi layer perceptron.

I. Introduction

In recent years many attempts have been made to predict the behavior of bonds, currencies, stocks, stock markets, or other economic markets [1]. These attempts were encouraged by various evidence that economic markets do not behave randomly, but rather perform in a chaotic manner [3]. A portfolio, arranged in the same ratio as the components of a stock market's index, provides the same performance as the index. It is therefore sufficient to model an index in order to describe the behavior of a market.

Several models can be used to predict the next day's value, the upcoming change, or just to give a binary rise-fall output. In this paper the latter approach is favored. The main focus is directed to maximizing the performance of the portfolio rather than maximizing the percentage of correct decisions. A single lost chance may hurt more than several negligible missed opportunities.

II. Data

The input data for both networks consists of market data collected between February 1994 and September 1995. Thus far, best results are obtained by using recent S&P Index values as well as a currency index, which is composed of the exchange rates of the three currencies Japanese Yen, British Pound and German Mark. Only the closing values are used. For the prediction of the next day's change, today's value, the change with respect to the five previous days, and the change from two weeks previous are investigated as possible inputs to the network. All inputs are normalized. Due to the nature of Stock markets, most changes are only of very small amplitude compared to the maximum changes. In order to receive a more equal distribution over the input space, a sigmoidal normalization function is used.

The full data set consists of more than 400 patterns. These patterns were divided into four equally sized groups. Three of these were used for training and the remaining is used for testing of the network.

III. Classification Model and Original Model

The networks are designed to give advice about whether to buy or to sell stocks. They can give three different outputs: "buy", "sell", and "keep current status". The last output is trained in case of minor changes in the value of the index (i.e. +/- 0.2%) to avoid unnecessary trading. These changes are too small to be considered in the process of decision making.

The performance measure of the network is a portfolio starting at a value of 100 and changing the same percentage as the index every time a "buy" advice is given. This portfolio is compared to the maximum possible performance and the performance of the index.

One of the two architectures used in this comparison is a multilayer perceptron which is trained with the backpropagation algorithm [4]. Experiments were conducted with one and two hidden layers. All layers have a sigmoid activation function, which in the case of the

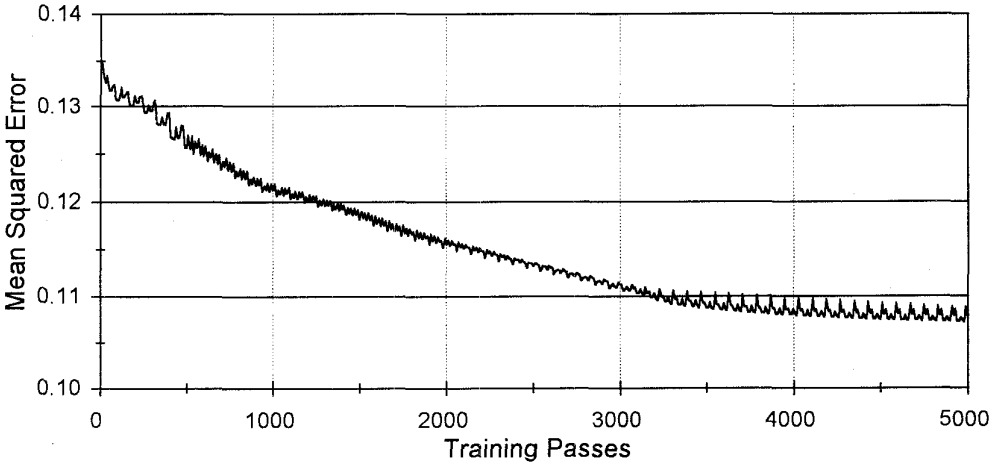

Figure 1: Mean Squared Error During Learning

input layer more evenly distributes the input values. The output layer consists of three nodes, each one corresponding to one of the possible decisions. This layer performs like a winner-take-all layer, i.e. the node with the highest activation is chosen as the advice of the network.

The mean squared error (MSE) of a three-layer perceptron (one hidden layer), as shown in Figure 1, seems to stay at a very high level. It can, though, be reduced further using more training passes. Values of less than 0.085 for the MSE and more than 75% correct classifications on the training set are reached after 20000 iterations. Unfortunately, this doesn't help at all, because the network starts "memorizing" the trained values and looses the ability to generalize based on unknown data. This phenomenon of overfitting is one of the major problems when dealing with this noisy data. The goal must be to stop training when the performance of the testing set, which is closely related to the percentage of correct decisions, reaches its maximum.

In Figure 2 the performance of the testing set (i.e. the final value of the portfolio) is shown with the percentage of correct decisions in the training set. While the first peak in the performance graph after 500-600 iterations is probably due to chance, there is a visible high between 1500 and 2500 training passes and a steady decline afterwards. This suggests that a very good generalizing ability is reached in an early stage of learning. Many experiments were made determining the best number of hidden neurons. Here again the impact on the generalizing ability was of great importance. Small numbers of nodes in the hidden layer (four or six) keep the network from learning anything. The complexity of patterns cannot be represented. Using more than 12 hidden neurons also results in overfitting: the network memorizes the patterns and cannot deal with unknown inputs.

Results of the performance of the net compared to the index and the maximum possible performance are shown in Figure 3. At this state, 1900 training passes were performed. The net manages to keep track with an inclining market and gains advantage over the index in an declining period. Most of the large losses are detected by giving a "sell" advice.

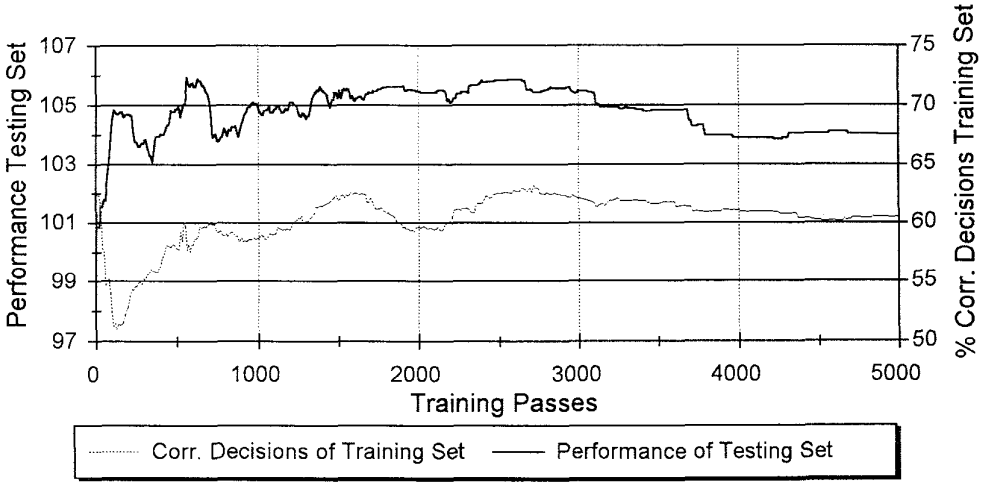

Figure 2: Performance of Testing Set

By introducing a second hidden layer in the perceptron architecture, the learning process, understood as minimizing the MSE, is speed up tremendously. This does not lead to a generally better performance, though. In this example, a network with 16 hidden neurons in the first, and eight hidden neurons in the second hidden layer was used. As shown in Figure 4, two areas of good performance are visible, one between 2000 and 2500 training passes, the other around 5400. Using more hidden neurons shifts the two peaks to earlier stages of learning. Figure 5 shows the performance of a four-layer perceptron after 5400 training passes.

An inherent difficulty of the presented classification approach using multi-layer perceptrons is the little differentiation of the data. Three classes might not be enough in order to emphasize on the important decisions, i.e. predicting especially the large changes of the

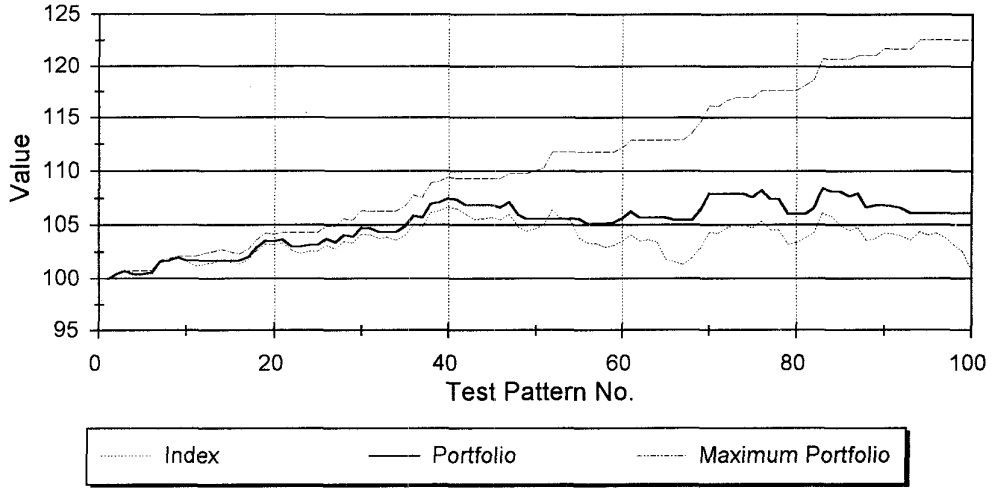

Figure 3: Performance of Testing Set after 1900 Training Passes

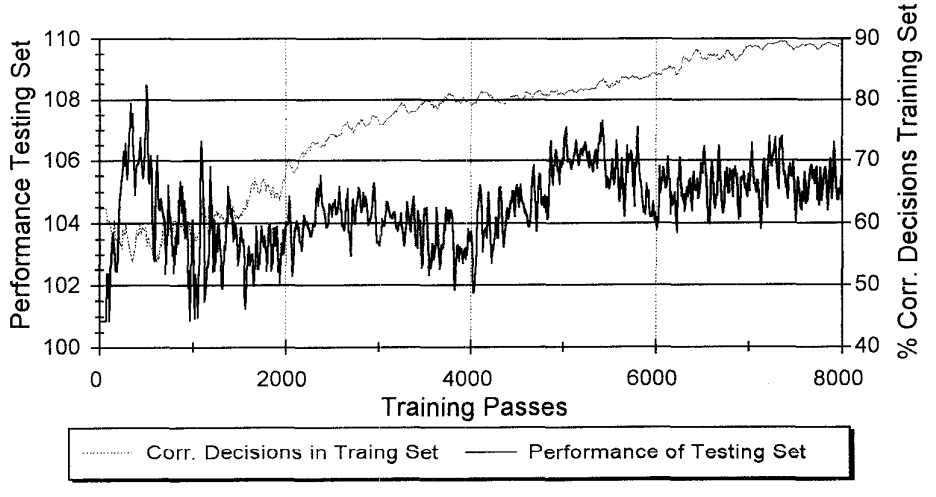

Figure 4: Performance of Testing Set Using Two Hidden Layers

index correctly. By using more classes, separate learning rates for each class or a more frequent training of patterns with large changes could be added to the algorithm. Nevertheless, such a change in architecture would result in an enormous increase in complexity, since the number of neurons in the output layer has to be changed and the network must be trained to perform an even more precise classification. This is one reason, why a probabilistic network might be a better approach to this problem.

IV. Proposed Model: The Probabilistic Neural Network

Several authors, including [6] and [2], have received good results by using Probabilistic Neural Networks (PNN) for stock market forecasting. Probabilistic Neural Networks, as proposed by Specht [5], perform a statistical classification task. For each category, a probability density function, which depends on the training patterns, is calculated. In this project, a Gaussian function is used as the Parzan Weighting function, such that the Probability function, evaluated for an input X, can be written as

$$f_A(X)=\frac{1}{(2\pi)^{p/2}\sigma^p}\frac{1}{m}\sum_{i=1}^{m}\exp\left(-\frac{(X-X_{Ai})^t(X-X_{Ai})}{2\sigma^2}\right)$$

where p is the number of dimensions of the input pattern X, and m is the number of training samples of category A. Each category i is then associated with an expected loss l_i, if the wrong category was chosen. The class θ_k finally to be chosen is determined using:

$$d(\boldsymbol{X})=\theta_k \text{ if } l_k f_k(\boldsymbol{X})>l_i f_i(\boldsymbol{X}) \;\forall i\neq k$$

The loss value allows an emphasis on decisions in extreme cases. For the stock index forecasting problem, not three but eleven classes are used. This enables the representation

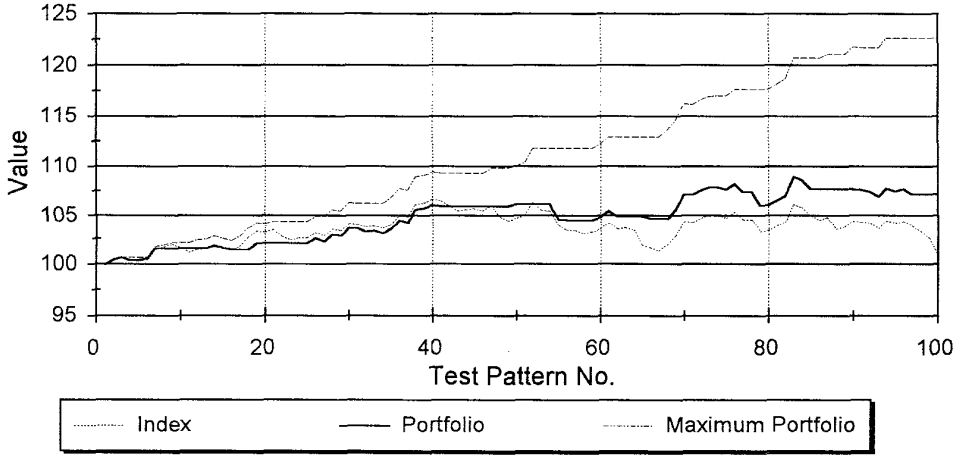

Figure 5: Performance of the Two-Hidden-Layer Network
after 5400 Training Passes

of different degrees of change. The loss values vary between 0.8 for the central and 2.0 for the extreme categories, which makes it more likely for the rarely occurring, extreme classes to be chosen. Besides the loss values, the only parameter to choose is the spread constant σ.

Figure 6 shows the same testing set as in Figure 5, this time using a PNN with σ=0.25. It is noticeable that the network detects almost all major losses of the index. In the first half of the testing period, where the market trend is going up, the network can hardly outperform the index. In the second half, the network still gains value while the index decreases by more than 5%. This forecasting behavior, which rather accepts lost opportunities than it risks high losses, is partially explained in the conclusions.

The optimal value for the spread constant, which was set manually in the above case, should be determined automatically by the program. For this task, a solution, mentioned in many papers such as [2], is implemented. The training patterns are divided into two groups, a training set and an evaluation set. While the network is trained using the training set, the evaluation set constantly determines the generalization ability of the present network. It can be assumed, that a network, fine-tuned to the classification of the not-learned evaluation set, will also do a reasonably good job classifying the unknown testing set.

The quality of generalization is measured by the percentage of correctly made decisions, where more important decisions are weighted high in a way that five small errors have the same negative effect on the performance as one major false prediction.

V. Concluding Remarks

For the given data set, in which the index rose from 100 to 100.86 and the maximum possible result is 125.66, the multilayer perceptron could perform to 106.10 with one hidden layer and slightly better to 107.21 by using two hidden layers, while the probabilistic neural

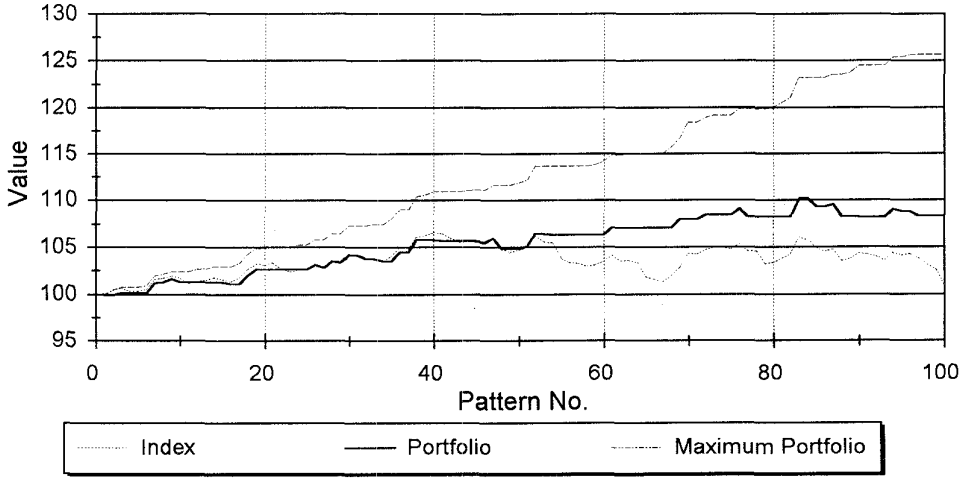

Figure 6: Performance of the PNN with spread value 0.25

network accomplished a final portfolio value of 108.30. The slightly better performance of the PNN was also found in some of the other test sets, which are not shown in the figures. Unfortunately, the automated optimization procedure for finding a well performing spread constant cannot guarantee to find the spread value for peak performance of the testing set. In fact, experiments show that an automated selection of σ leads to results which are up to 4% worse than the optimal values, depending on the data set chosen for the experiment. The very best spread value can only be determined after applying it to the testing set which is not available to the optimization procedure.

All tested architectures show a better performance relative to the index, when the index is decreasing. This is an inherent characteristic of the given problem structure: in a declining market, the portfolio can only perform better than or equal to the index. To beat the index during an inclining market, not only the increasing steps have to be predicted correctly, to keep track with the index, but also small declines must be detected to gain advantage over the index. The latter is obviously much more demanding.

At this time it would be too early to generally favour the stock market forcasting ability of a probabilistic neural network over a multilayer perceptron, but there are several advantages of the PNN approach. The multilayer perceptron in the present implementation does not pay special attention to the importance of possible high magnitudes of change in the stock market index. Three classes may not be sufficient to model this problem. Furthermore, due to their exclusivity [7], the PNN seems to be suited better to perform the classification task over a highly separated input space.

Most important, a multilayer perceptron usually works in an static environment with separate training and testing phases. The stock market, on the other hand, is a fast changing environment, that calls for an adaptive approach for forecasting the change. At this point, no adaptivity is built into the above presented architectures, but due to its one-step learning, this characteristic can be easily implemented in a PNN.

Nevertheless, other neural network architectures such as radial basis functions should be investigated for their classification and forecasting performance.

VI. References

[1] Chakraborty, Kanad; Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka (1992). Forecasting the Behavior of Multivariate Time Series Using Neural Networks. *Neural Networks*, **5**, 961-970.

[2] Chen, C.H. (1994). Neural Networks for Financial Market Prediction. *Proc. 1994 IEEE International Conference on Neural Networks*, 1199-1202

[3] Malliaris, Mary E. (1994). Modeling the Behavior of the S&P500 Index: A Neural Network Approach. *Proc. 10th IEEE Conference on AI for Applications*, 86-90.

[4] Rummelhart, David E., and James L. McClelland (1986). *Parallel Distributed Processing*, **1**, 318-362.

[5] Specht, Donald. F. (1990). Probabilistic Neural Networks. *Neural Networks*, **1**, 109-118.

[6] Tan, Hong; Danil V. Prokhorov, and Donald C. Wunsch II (1995). Probabilistic and Time-Delay Neural Network Techniques for Conservative Short-Term Stock Trend Prediction. *Proc. World Congress on Neural Networks*, **II**, 44-47.

[7] Werbos, Paul (1992). Neurocontrol and Supervised Learning: An Overview and Evaluation. In: White, David A., and Donald A. Sofge, ed.: *Handbook of Intelligent Control.* Van Nostrand Reinhold, 65-89.