Selection of Learning Algorithms for Trading Systems Based on Biased Estimators

Zoran Obradović[1,*] Tim Chenoweth[1,2,3]

zoran@eecs.wsu.edu tchenowe@eecs.wsu.edu

[1] School of Electrical Engineering and Computer Science

[2] Department of Management and Systems

[3] Department of Economics

Washington State University, Pullman WA 99164-2752

Phone: 509-335-6601, Fax: 509-335-3818

Abstract

In our previous trading system for the S&P 500 index, promising results were obtained by partitioning the historic data into disjoint subsets used to design two biased local estimators whose partial estimates were combined into a trading recommendation [3, 4]. The objective of this study is to explore whether using cascade-correlation learning instead or in addition to the previously used back-propagation to train either one or both of the local estimators improves the trading system's performance. Several learning algorithm combinations were explored and tested using real financial data. The system yielding the best results used a mixture of learning algorithms (both back-propagation and cascade-correlation) and achieved an annual rate of return of 20.49% without a commission and 14.37% with a 0.05% commission over a five year trading period. This is significantly better than the annual rate of return achieved by both the buy and hold strategy (13.36%) and a system configuration that did not use the cascade-correlation algorithm (11.54% with no commission, 5.35% with a 0.05% commission).

*Research sponsored in part by the NSF research grant NSF-IRI-9308523.

1. Introduction

The prediction of future returns for financial markets is a highly complex time series problem that is receiving much attention from researchers in industry [10] and academia [14]. In general, financial markets are highly efficient, which means there is little relationship between historical data and future market returns. Those relationships that do exist disappear rapidly once they are discovered. This is especially true when linear relationships are involved because most traditional market forecasting techniques rely on linear modeling and are capable of identifying such linear relationships in the data [1]. However, it may be the case that nonlinear relationships exist that are overlooked by simple linear models. A nice characteristic of nonparametric machine learning techniques is their ability to discover existing nonlinear relationships automatically and directly from provided historical data [12]. One possible learning method is based on neural networks, which are powerful computational structures that can theoretically approximate almost any nonlinear continuous function on a compact domain to any degree of accuracy [6]. Previous published neural network research related to this study includes an attempt to build several financial time series prediction models using various measures [9], some of which are used in our work. Another related work uses a *gating network* to integrate the results from several local experts for specific market conditions [11].

This study extends our previous neural network based trading system described in [3, 4]. Our approach partitions the prediction problem into two subproblems, models each subproblem separately, and combines the estimates into a final prediction. This partitioning is accomplished by separating the training set into disjoint subsets that are used to train neural networks, which act as *local estimators*. The local estimates from both neural networks are integrated into a single prediction by the combining algorithm, which suggests a trade. Training the local estimators on disjoint data sets (as opposed to data sets that overlap) creates less correlated local estimates, resulting in improved system performance.

In our previous work, local estimators were prespecified neural networks employing back - propagation learning [13]. Although this simplified model assumes that the proposed problem partitioning results in subproblems of similar complexity and that the problem complexity is not changing significantly over time, the obtained results were encouraging. This paper extends the previous model by allowing the system to be further improved through the use of the cascade-correlation learning algorithm [7], which can vary the neural network size for each prediction step. The study also analyzes whether a mixture of neural networks employing both back-propagation and cascade-correlation learning could further reduce the correlation between the local estimators, yielding a better trade recommendation.

A primary advantage of cascade-correlation learning is that the algorithm does not require exhaustive experimentation in order to specify an appropriate number of neural network computational units. The algorithm adds computational units one at a time until a stopping criteria is reached. This improves the model's flexibility since it allows the number of model parameters to vary in accordance to the problem's changing distribution. In contrast, the back-propagation learning algorithm requires that the number of computational units be prespecified. Additionally, a neural network using the cascade-correlation algorithm is trained faster than one using back-propagation.

In summary, the objective of this paper is to determine whether the cascade-correlation algorithm can be employed in a manner that improves our trading system's performance. Results from this study are compared to our previous results and to the buy and hold strategy. A brief overview of relevant neural network terminology and ideas is provided in Section 2, followed by system implementation details explained in Section 3. Experimental results are discussed in Section 4 and conclusions are presented in Section 5.

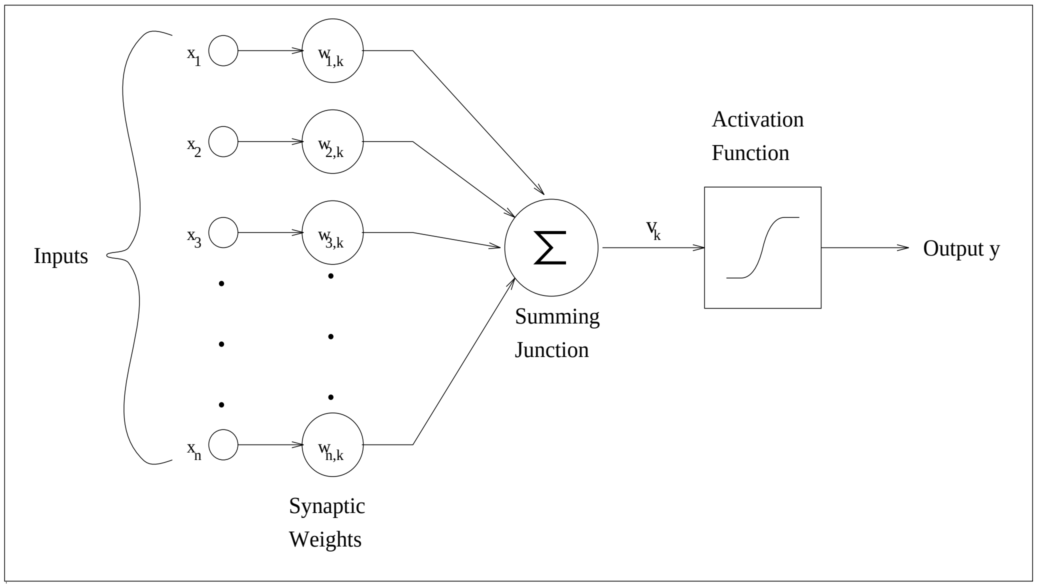

Figure 1: Model of a Neuron.

2. Background

A neural network is a parallel computational structure composed of relatively simple interconnected computational units (*neurons*). A neural network acquires its knowledge through a learning process according to a learning algorithm. The knowledge gained in this manner is distributed throughout the net by modifying the strength of the interconnection links (*weights*).

An individual neuron (shown in Figure 1) computes the weighted sum of inputs as

$$v_k = \sum_{j=1}^{n} w_{j,k} x_j, \tag{1}$$

where the x_j's are the inputs to neuron k, and the $w_{j,k}$'s are the corresponding connection weights from neuron j to neuron k. The output of neuron k then becomes

$$y_k = \varphi(v_k - \theta_k), \tag{2}$$

where θ_k is called the *threshold* and the function φ is called the *activation* function.

The activation is typically a nonlinear function that limits the output of a neuron to a specific range and gives the network the capability of modeling nonlinear functions. Commonly used

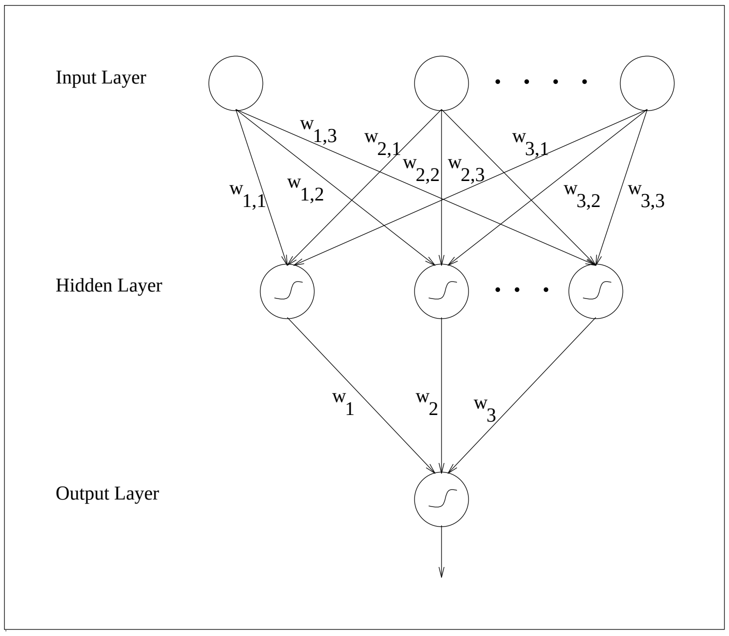

Figure 2: Multilayer Feedforward Network Architecture.

activation functions are the *sigmoid function* defined as

$$\varphi(v_k) = \frac{1}{1 + e^{-v_k}} \tag{3}$$

and the *hyperbolic tangent function* defined as

$$\varphi(v_k) = \frac{1 - e^{-2v_k}}{1 + e^{-2v_k}}. \tag{4}$$

The sigmoid function maps the output of a neuron to the $(0, 1)$ range, whereas the hyperbolic tangent function maps it to the $(-1, 1)$ range.

The structure of neurons in a neural network is often referred to as the network *architecture*. The multilayer *feedforward neural network* architecture shown in Figure 2 is a popular choice in the literature. The main characteristic of this architecture is an acyclic interconnection graph with one or more *hidden layers* of neurons. These hidden layers allow the network to extract higher order statistics from the training data set, enabling the network to model unknown nonlinear functions. Neurons in the input layer supply the individual components (*features*) of the input to the neurons in the first hidden layer. The output from the neurons in the first hidden layer becomes the input to the neurons in the following layer. The output from the neurons in this layer becomes the input to the neurons in the next layer and so on, from layer to layer, throughout the entire neural network. The output from the neurons in the output layer is the network response to the specified input. Feedforward neural networks are generally *fully connected*, meaning that every neuron in a specific layer is connected to every neuron in the previous layer.

A key characteristic of a neural network model is the ability to adapt to its environment through *learning*. This is an iterative process accomplished through repeatedly changing the network's weights. A learning algorithm thus becomes a sequence of steps that alter the network's weights in order to improve the model's performance on a given *training data set*. Once learning is completed the neural network is generally presented with a test example and a corresponding *prediction* computed using the final network configuration. Two well known learning algorithms used in this study, called back-propagation [13] and cascade-correlation [7], are briefly described in the following sections. Practical details concerning neural networks in general and both algorithms in particular may be found in [8].

2.1. The Back-Propagation Learning Algorithm

The back-propagation method computes the prediction error for a particular input and attempts to minimize it by adjusting the neural network weights according to the gradient descent optimization principle. More precisely, each *training example* is an input-output pair $(\vec{p}, d(\vec{p}))$ from an unknown function, which we want to approximate using a given *training* set of such examples. For a given input $\vec{p}$, the desired output $d(\vec{p})$ is compared to the network's output $y(\vec{p})$ and an *error* value is computed as

$$e(\vec{p}) = d(\vec{p}) - y(\vec{p}). \tag{5}$$

Back-propagation learning is a process of adjusting the neural network weights in a manner that minimizes the sum of squared errors on the training set examples. Each weight adjustment step distributes responsibility for the error to all neurons through appropriate modification of their corresponding weights. Once all network weight corrections are computed for one training example, the corrections are added to the corresponding weights and the process is repeated using the next example. A complete pass through the training set is called an *epoch*, and a training session generally lasts several thousand epochs. The stopping criteria for the back-propagation learning algorithm is either the average error over all training examples falling below some minimum error value, or reaching the maximum allowable number of epochs.

The objective of the back-propagation algorithm is to optimize the weight parameters in a prespecified fixed neural network architecture. This means that the number of neurons and the interconnection graph is prespecified and does not change during the learning process. In this study, back-propagation learning is applied to single hidden layer feedforward neural networks with a prespecified number of neurons determined by a trail and error process.

2.2. The Cascade-Correlation Learning Algorithm

Another learning algorithm employed in this paper is cascade-correlation [7]. This algorithm utilizes the *cascade architecture* which, in contrast to the architecture used by the back-propagation algorithm, starts without any hidden units and adds them dynamically during the training process. The cascade-correlation training algorithm is also totally different from the back-propagation algorithm. Although the method is still gradient descent based, connections to the hidden units are trained in a manner that maximizes the correlation between the hidden units' output and the error value for the output unit. The connections to the output unit are trained as in back-propagation.

Cascade-correlation learning starts with an architecture that has only input and output units (see Figure 3). The neural network is trained for a number of epochs and the weights are adjusted using gradient descent optimization to minimize the sum of squared errors as described in the

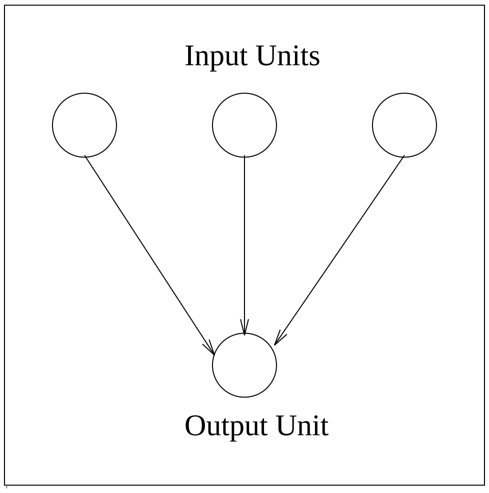

Figure 3: Initial Cascade Architecture.

previous section. When the network is trained (a stopping criterion is satisfied) an additional pass through the training set is made and an error value is computed for each training example. Next, a hidden unit is added to the network as shown in Figure 4. Initially, the weights corresponding to the new connections marked with a square in Figure 4 are trained to maximize the correlation between the output of the hidden unit and the corresponding error value for the output unit. This local optimization problem is solved by adjusting only the input-to-hidden unit weights using the gradient descent technique over a number of epochs. The connections to the new hidden unit are fixed after this local training process for the remainder of the training session. Now, the weights corresponding to the connections leading to the output unit (connections marked with a circle in Figure 4) are trained again to minimize the sum of squared errors.

Next, an additional hidden unit is added to the network with the resulting architecture shown in Figure 5, and the weight optimization process is repeated. The new weights marked with a square in Figure 5 are trained to maximize the correlation between the new hidden unit's output and the error value for the output unit. Then, the weights marked with a circle are trained to minimize the sum of squared errors. The weights marked with an X are fixed and do not change during the remainder of the training session. Additional hidden units are added to the network until some stopping criterion is satisfied and the training session ends. The stopping criterion used in this

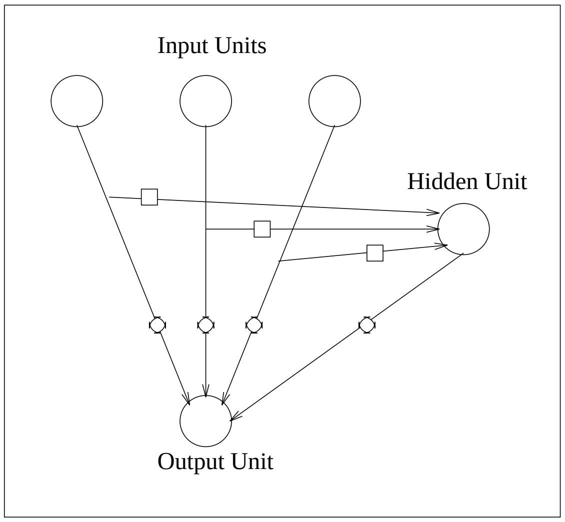

Figure 4: Cascade Architecture with Hidden Unit.

paper for the cascade-correlation learning algorithm is either the sum of squared errors averaged over all training examples falls below some minimum error value, the correlation between the new hidden unit's output and the error value of the output unit falls bellow some minimum correlation value, or that the maximum allowable number of hidden units is reached.

In theory, the cascade-correlation algorithm is computationally more efficient than the back-propagation learning algorithm. Each neuron in a back-propagation network uses an iterative approach to adjust the weights leading to it in a manner that will minimize the sum of squared errors. However, all the neurons in the network are adjusting their weights at the same time, making the minimization problem quite complex. The cascade-correlation algorithm addresses this problem by training only part of its weights at a time, keeping the rest of the parameters fixed. As each new hidden unit is added to the network, the weights leading to the unit are trained, while the rest of the network weights are kept constant. When the weights leading to the unit are satisfactorily trained, they are fixed for the remainder of the training session. The only network weights that are retrained each cycle are those leading to the output unit.

A second disadvantage of the back-propagation algorithm is that the network architecture must

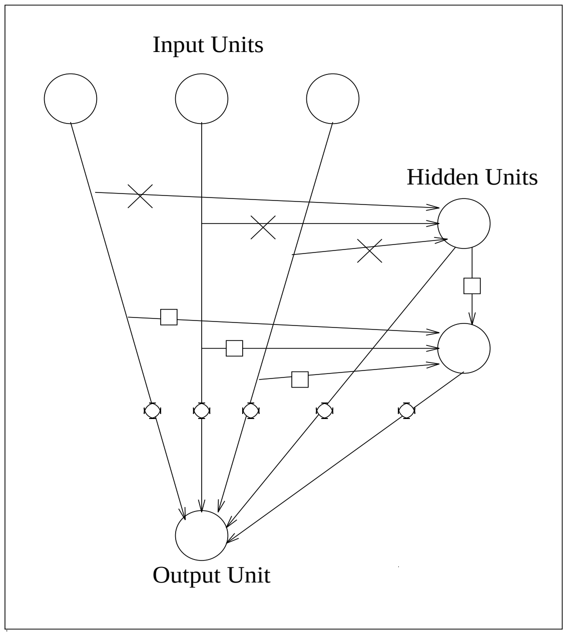

Figure 5: Cascade Architecture with Two Hidden Units.

be completely fixed at the beginning of the session. This is a serious problem, since there is no systematic and efficient approach for determining an optimal network architecture for a specific problem. Given appropriate stopping criteria, the cascade-correlation algorithm addresses this issue by allowing the network architecture to grow to whatever level of complexity is necessary to address the current problem. The initial architecture is extremely simple, with no hidden units. Each iteration adds a single unit to the system, increasing the number of system parameters and the computational power of the network. The architecture stops growing when the network has enough computational power to address the problem.

It is very important, however, that this growth be controlled through the proper selection of stopping criteria. If the cascade architecture is allowed to grow too large, *overfitting* may become a problem. Overfitting means that the neural network memorizes the training set data and looses its generalization ability. This results in poor predictions when the network is presented with an example that was not used in the training process. Overfitting is easier to control for neural networks using back-propagation learning since the number of system parameters can be fixed at a level that makes memorizing the training set very difficult.

3. Methodology

The trading system discussed in this study uses a limited amount of historically ordered training examples in order to design a model that provides the trading recommendation for the current day based on the predicted return for the following day. When the actual return for the following day becomes know, it is added to the training set and the oldest training example is removed. The modified training set is now used to design a new model, which is used to determine a new trade recommendation. This process of shifting the training window forward and designing a new model by learning from more recent examples is repeated each day. Designing a new model for each prediction step is much more computationally expensive than designing a model once and using it for an extended period. However, our experience suggests that the proposed methodology provides better results when dealing with a nonstationary time series with a changing distribution such as the S&P 500 index.

The trading system used in this study is composed of three components. The first is a *preprocessing component* that is used to perform feature selection and data filtering. Feature selection identifies the most informative data, while the data filter splits the training data into three disjoint sets, a positively biased set, a negatively biased set, and a noise set that is discarded.

The *learning component*, composed of two neural networks, is used to perform a prediction process. One network is trained using the positively biased training set, whereas the other is biased using the negatively biased set. The predictions from each neural network are combined into a system prediction by the *postprocessing component.* This component is composed of a single neural network that takes as input the predictions of each of the biased networks and combines these into a single system prediction.

3.1. System Description

The feature selection process, described in detail in [5], is necessary because the gradient descent based neural networks used in this study are computationally very expensive. In addition, they become significantly more expensive as the size of the input vector increases. Another consideration is the number of parameters in the model. More features used as input to the network result in more parameters in the model. To properly fit more parameters, more data is needed for neural network training. Therefore, to design a good and practical trading system it is important to use a manageable number of informative input features. The goal of *feature selection* is to reduce the set of candidate features to a manageable size in a manner which minimizes predictive information loss.

The feature selection technique used in our trading system consisted of combining the results from several different statistically based methods into a final feature set. This reduces the instability problems described by Breiman in [2]. Breiman pointed out that small changes in the data set used in the feature selection process can cause drastic changes in the final set of features. To minimize this effect for multivariate financial time series, we suggested a procedure which averages the results of several feature selection methods.

With our method, all the candidate features are ranked according to a specific selection technique and criterion and assigned a score based on this ordering (a score of one being the best). The procedure is repeated several times using different combinations of selection techniques and criteria. The scores for each combination are summed, providing a final score for each feature. A final feature set with cardinality n is then determined as the n features with the lowest scores. The initial candidate feature set for the S&P 500 index predictions contained 67 features, which was reduced by the preprocessing component to the feature set used in this study, containing only the 6 features shown in Table 1.

The data filter partitions the training set into three disjoint sets. The first set, containing

S&P 500 index return
S&P 500 index return lagged one day
S&P 500 index return lagged two days
U.S Treasure Rate lagged 2 months
U.S Treasure Rate lagged 3 months
30 Year Government Bond Rate

Table 1: Features of Each Example.

training examples corresponding to large positive returns, is used to train our *optimistically biased neural network*. The second data set, containing examples corresponding to large negative returns, is used to train the *pessimistically biased neural network*. The third data set, containing the remaining examples, is discarded as noise (see [4] for details). The set to which a specific training example is assigned depends on the magnitude of the corresponding S&P 500 index return. The desired output ($d(\vec{p})$, defined in Section 2) is the actual return value for the next day. This value is compared to a predefined threshold h and if $d(\vec{p})$ is greater than h, training example $(\vec{p}, d(\vec{p}))$ is added to the training set for the positively biased neural network. If $d(\vec{p})$ is less than $-h$, the training example is added to the training set for the negatively biased neural network. Finally, if $d(\vec{p})$ is between $-h$ and h ($-h < d(\vec{p}) < h$), the training example is discarded as noise. For this study, h was experimentally set to 0.5% as explained in [4].

The motivation for partitioning the training set in this manner is to create two biased local estimators with the correlation between these two estimators as small as possible. Such a design with local estimators trained on disjoint data sets (as opposed to data sets that overlap) results in less correlated local estimates provided to the post processing component and hopefully in a better trading performance. The local estimators used in our system are the two neural networks. One network is trained using the positively biased training set and as such has an *optimistic view* of the market, while the other network is trained using the negatively biased training set and therefore has a *pessimistic view* for the market.

The postprocessing component of our trading system is a small neural network that takes the

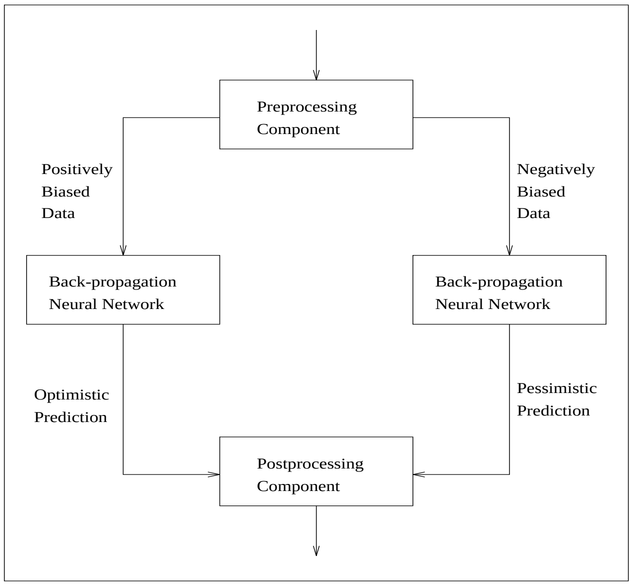

Figure 6: System A Structure.

biased predictions from each local estimator and combines them into a single prediction, which is used to make a trade recommendation. The postprocessor neural network has two input units, two hidden units, and one output unit and uses the back-propagation learning algorithm. The system trade recommendation depends on the output of the combiner neural networks. If the output is negative, the trade recommendation is to establish a *short market position.* If it is positive the recommendation is to establish a *long market position.* A long position means purchasing an asset for later resale, while a short position means selling a borrowed asset now and purchasing it later.

3.2. Learning Component Structure

Our previous trading system (System A having a structure as shown in Figure 6) employed neural networks using the back-propagation algorithm for both local estimators [3, 4]. Experiments were conducted over a short time period to determine the number of hidden units for each local estimator, which was then fixed throughout the global trading experiments over a much longer time period. A perceived problem with this system architecture was that the number of modeling parameters was determined at the beginning of the trading process by fixing the number of neural network

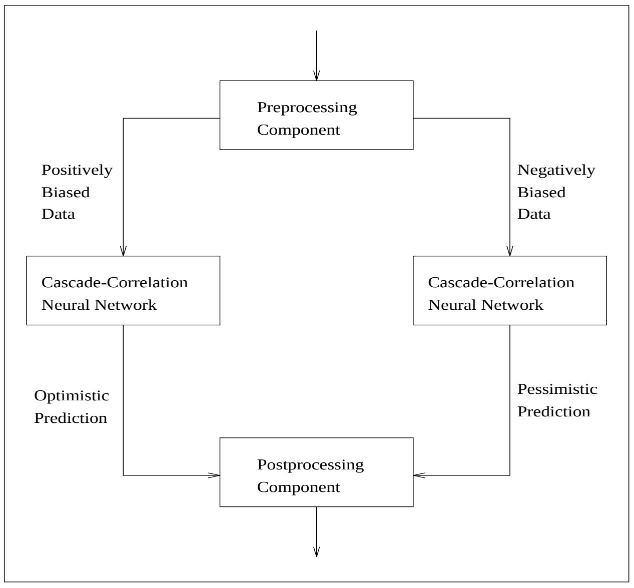

Figure 7: System B Structure.

hidden units. It would make more sense to allow these parameters to fluctuate according to the complexity of the problem. The objective of this study is to explore whether it is possible to further improve the results by using neural networks that employ the cascade-correlation learning algorithm. In particular, this study explores whether an architecture such as System B, shown in Figure 7, using cascade-correlation local estimators will further improve the trading results by reducing the correlation between estimators. As described in Section 2, the number of parameters in a network using the cascade-correlation learning algorithm is not fixed, but can grow to meet the demands of the current problem. This gives cascade-correlation based networks an advantage over the back-propagation based networks, which have a fixed number of parameters. It is also possible that a mixed configuration (for instance, a back-propagation network for the optimistic estimator and a cascade-correlation network for the pessimistic estimator) could improve results by further reducing the correlation between the biased local estimators. Figures 8 and 9 show the mixed structure systems C and D tested in this study.

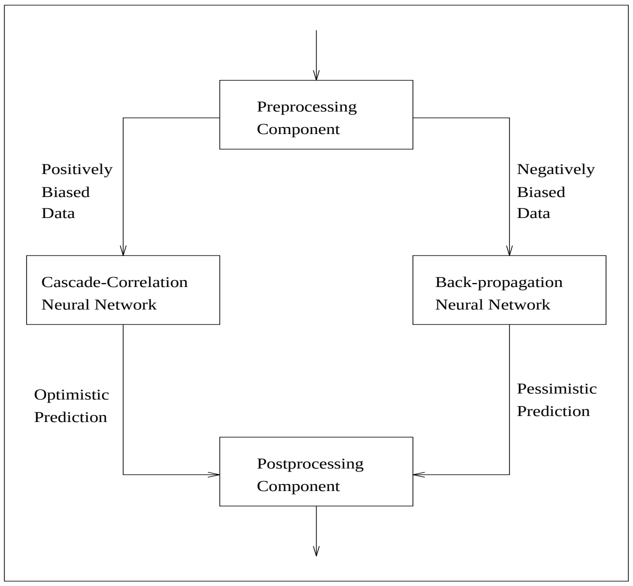

Figure 8: System C Structure.

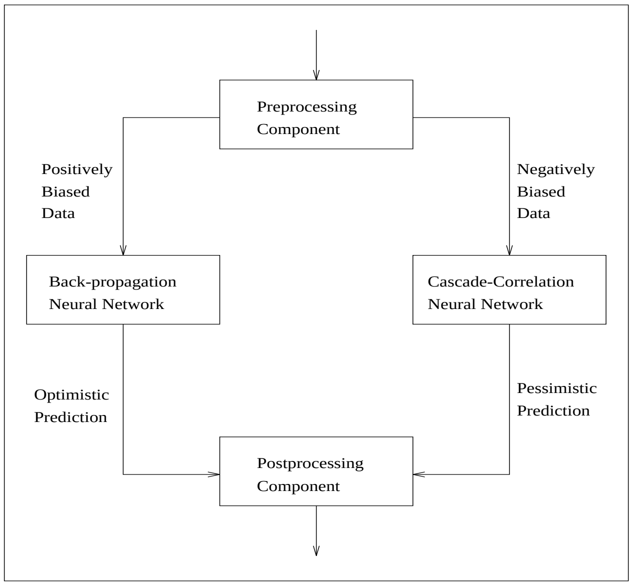

Figure 9: System D Structure.

3.3. System Performance Measures

The primary performance measure for a trading system is its *annual rate of return* (ARR). As such, the system's compounded annual rate of return is computed as

$$ARR = 100 \cdot \frac{K}{T}\left(\prod_{t=1}^{T}(1 + s_t r_t) - 1\right), \tag{6}$$

where T is the total number of experimental trading days, K is the number of trading days per year ($K = 253$ days for this study), r_t is the actual return for day t, and s_t is the trading signal for day t. The trading signal has two possible values: $s_t = 1$ means the trading system recommends a long position, while $s_t = -1$, means the trading system recommends a short position.

It is also important, when measuring the performance of a trading system, to consider *commissions*. These are costs incurred when a trade is actually executed. A trade occurs whenever s_t changes its value. For this study, the ARR is computed for transaction costs equal to 0%, 0.01%, and 0.05% per trade.

An additional measure reported in this study is the *overall market direction correctness* (MDC) of the system, computed as

$$MDC = 100 \cdot \frac{C}{T}, \tag{7}$$

where C is the number of trading days in which the system correctly predicted the market direction.

4. Results and Analysis

The data used for making S&P 500 trade recommendations is an ordered, daily, financial multivariate times series from January 2, 1982 to December 31, 1993. The initial training set contains the data through December 31, 1988, whereas daily trade recommendations are made on data from January 3, 1989 to December 31, 1993. Each recommendation is obtained by training the system using the most recent five years of data. The measurements used for each experiment are the

Parameter	*Value*
Training Window Size	2000
Activation Function	Hyperbolic Tangent
Network Topology	6-3-1
Learning Rate	0.05
Number of Epochs	5000

Table 2: Configuration Parameters for Back-Propagation Based Local Experts.

Parameter	*Value*
Training Window Size	2000
Activation Function	Hyperbolic Tangent
Input Units	6
Learning Rate–Output Unit	0.01
Learning Rate–Hidden Unit	0.05
Training Set MDC	at most 55%
Training Examples per Parameter	at least 4
Number of Epochs per Neuron	500

Table 3: Configuration Parameters for Cascade-Correlation Based Local Experts.

compound annual rate of return assuming no commission, a 0.01% commission, and a 0.05% commission. The total number of trades and the overall market direction correctness are also reported. The results are compared to those achieved for the same period using the buy and hold strategy, a single back-propagation neural network, and a single cascade-correlation network.

The configuration parameters for back-propagation based local estimators is shown in Table 2. The network topology gives the explicit network configuration, with 6-3-1 corresponding to a network with six input units, a single hidden layer with three units and a single output unit. The initial training window contains 2000 examples, but it is important to note that this training set is partitioned into three disjoint sets as described in Section 3. On average, this partitioning resulted in 1000 examples being discarded, about 450 examples in the negatively biased training set and 550 examples in the positively biased set. The stopping criterion employed for these back-propagation networks was 5000 epochs.

The cascade-correlation based local estimators used the configuration parameters shown in Table 3. Two learning rates were used because the optimization problem of training the connections to

Parameter	*Value*
Training Window Size	2000
Activation Function	Hyperbolic Tangent
Network Topology	2-2-1
Learning Rate	0.01
Number of Epochs	1000

Table 4: Configuration Parameters for Combiner Neural Network.

the output unit (minimizing the sum of squared errors) is different than the optimization problem of training the connections to a hidden unit (maximizing the correlation between the hidden unit's output and the network error). The overall market direction correctness on the training set (55%) was used as a stopping criterion. After a new hidden unit is added, the *MDC* for the training set is computed and if it is greater than the predetermined maximum, the learning process is halted in order to avoid overfitting the data. Another stopping criterion was the minimum number of training examples per parameter (set to 4 in this study), which places an upper bound on the number of hidden units that may be added to the network. Before a hidden unit is added to the network, the total number of network parameters (weights) is computed and the ratio of the number of training examples to the number of network parameters is determined. If this ratio is smaller than the predetermined minimum, the training process is halted. Each time the connections to either the output unit or to a hidden unit are trained, the training lasts for a prespecified number of epochs (500 epochs in our experiments).

The configuration parameters for the combiner neural network, which used the back-propagation learning algorithm in all our experiments, is shown in Table 4. It can be observed that the smaller combiner neural network is trained using much more data than is available to the corresponding larger local estimators. Consequently, to improve system efficiency the number of epochs for the combiner neural network was reduced to 1000.

The experimental results are shown in Table 5. The first column in Table 5 assigns a letter to each system architecture. The second column describes the estimator structure. The subcolumn

System	*Estimators*		*ARR for various commissions (c)*			*MDC*	*Trades*
	Optimistic	*Pessimistic*	$c = 0.00\%$	$c = 0.01\%$	$c = 0.05\%$		
A	Back-Prop	Back-Prop	11.54%	10.31%	5.35%	50.20%	527
B	Cascade	Cascade	0.13%	-1.24%	-6.76%	51.45%	697
C	Cascade	Back-Prop	-5.23%	-6.41%	-11.12%	49.33%	711
D	**Back-Prop**	**Cascade**	**20.49%**	**19.26%**	**14.37%**	**51.85%**	**485**
E	Single Back-Prop Estimator		-2.57%	-3.07%	-5.10%	48.15%	261
F	Single Cascade Estimator		20.01%	17.73%	8.63%	52.79%	933
G	Buy & Hold		13.36%			52.71%	2

Table 5: Experimental Results.

marked *Optimistic* shows the learning algorithm employed by the optimistically biased local estimator, whereas the subcolumn marked *Pessimistic* shows the learning algorithm employed by the pessimistically biased local estimator. Systems E and F are single global estimators trained using back-propagation and cascade-correlation learning, respectively. For these two experiments, the training data was partitioned as discussed previously and the noise set was discarded. However, the positively biased and the negatively biased training sets were combined and used to train a single neural network with the specified learning algorithm. The prediction from this single network was used to determine the trade recommendation. The final system (System G) is the simple *buy and hold strategy*, which means that the index (S&P 500 in this study) is bought at the beginning of the period and sold at the end (five years later). Another perspective to the annualized rate of return for different commissions is shown in Figure 10. The horizontal line in Figure 10 marks the return achieved using the buy and hold strategy.

The best results from the perspective of the *ARR* metric were achieved by system D, outperforming the buy and hold strategy under all three commission assumptions. However, it is important to observe that the other mixed configuration (system C) had the lowest *ARR*. It is also interesting to note that system F achieved an *ARR* almost as high as system D under the zero percent commissions assumption, and it was the only system with a percentage correct score higher than the buy and hold strategy score. However, the results of system F were greatly reduced under more

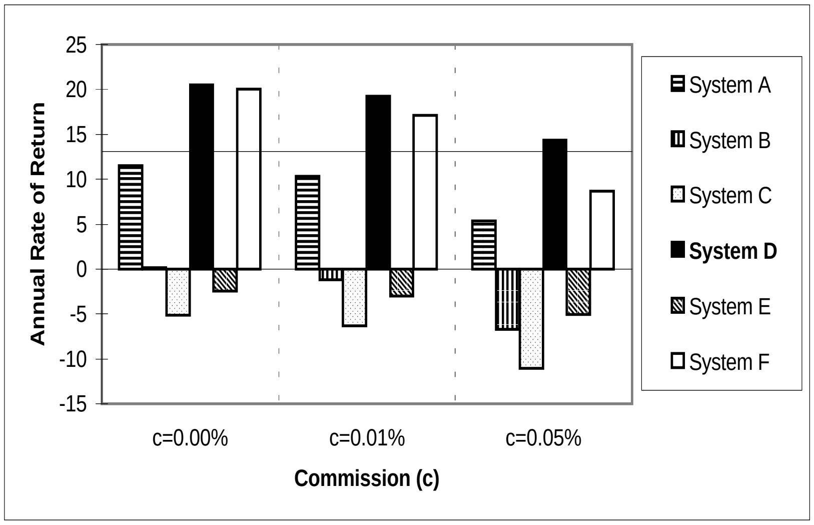

Figure 10: System Returns Versus Commission.

realistic assumption of paying a commission per trade since the system required a large number of trades.

Also, observe that the cascade-correlation based global estimator F outperformed the back-propagation based global estimator E. Additionally, a back-propagation based model performed better when used as the optimistic local estimator, while a cascade-correlation based model performed better as the pessimistic local estimator. This may be due to our problem partitioning creating subproblems of different complexity. While the subproblems are similar in size, their underlying probability distribution may be quite different, with the optimistically biased subproblem being significantly easier. If so, a cascade-correlation based model that performs well on one subproblem may not perform well on the other without first modifying the stopping criteria. It appears the the cascade-correlation stopping criteria used in this study results in a larger network (on average eight hidden units resulting in 90 weighted connections) that works well when used as the pessimistic local estimator, but shows overfit problems as the optimistic local estimator. In contrast, the prespecified architecture for the back-propagation based model was a network with 21 weighted connections, which seems to be more appropriate when used as the optimistic local estimator.

5. Conclusion

This study explored several trading systems, each of which partitions the prediction problem into two subproblems, models each subproblem separately using biased local estimators, and combines the estimates into a final prediction. Systems which produced an annual rate of return larger than the buy and hold strategy did not have larger overall market direction correctness scores. The system with *ARRs* larger than the buy and hold strategy obtained a better performance by correctly predicting the larger market movements. This suggests that the best strategy for trading systems might be to focus on correctly identify major market movements instead of trying to achieve a greater overall market direction correctness than that achieved by the buy and hold strategy.

The results indicate that experimentation is very important in determining the appropriate structure of a trading system based on biased estimators. In this study, a cascade-correlation based model performed better than a back-propagation based model when used as a pessimistic local estimator, but a back-propagation model was more appropriate as an optimistic local estimator. The best results were achieved using a mixture of learning algorithms (back-propagation for the optimistic and cascade-correlation for the pessimistic local estimator).

The main point of this paper is not that one algorithm is better than the other. With more exhaustive experimentation one should be able to discover more appropriate stopping criteria for the cascade-correlation based optimistic local estimator and a more a appropriate architecture for the back-propagation based pessimistic estimator. However, when modeling large complex problems, finding the optimal parameters is generally not computationally feasible. Therefore it is of practical interest to identify a reasonably good model through limited experimentation on several systems with prespecified learning parameters, as proposed in this study.

It is likely that the results of the systems employing biased local estimators can be improved further by modifying the manner in which the estimates are combined. The combiner neural network takes the two local estimates as inputs and learns how much weight to assign to each when determining the final prediction. Our research in progress focuses on providing the combiner network with context information in the form of additional inputs that could potentially improve the system's final prediction.

Acknowledgments

We would like to thank Radu Drossu and Pedro Romero for their constructive comments on the preliminary version of this manuscript.

References

[1] Black F. and Scholes M., "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, Vol. 81, May-June, 1973.

[2] Breiman L., "The Heuristics of Instability in Model Selection," Technical Report No. 416, Statistics Department, University of California, Berkeley.

[3] Chenoweth T., Obradovic Z., and Lee S., "Embedding Technical Analysis into Neural Networks Based Trading Systems," *Applied Artificial Intelligence Journal*, in press.

[4] Chenoweth T. and Obradovic Z., "A Multi-Component Nonlinear Prediction System for the S&P 500 Index," *Neurocomputing Journal*, in press.

[5] Chenoweth T. and Obradovic Z., "An Explicit Feature Selection Strategy for Predictive Models of the S&P 500 Index," *Neurove$t Journal,* Vol 3, No. 6, November 1995, pp. 14-21.

[6] Cybenko G., "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, 2 (1989) pp. 303-314.

[7] Fahlman S. and Lebiere C., "The Cascade-Correlation Learning Architecture," *Advances in Neural Information Processing Systems 2*, pp. 524-532, 1990, San Mateo CA.

[8] Haykin, S., *Neural Networks: A Comprehensive Foundation,* Macmillan College Publishing Company, Inc., 1994.

[9] Hutchinson J., Lo A., Poggio T., "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks," Technical Report, A.I. Memo No. 1471, C.B.C.L. Paper No. 92, Massachusetts Institute of Technology, April 1994.

[10] Mahfoud S. and Mani G., "Genetic Algorithms for Predicting Individual Stock Performance," *Proc. Third Int. Conf. on Artificial Intelligence Applications on Wall Street*, New York, NY, 1995, pp. 174-181.

[11] Weigend A. and Mangeas M., "Avoiding Overfitting by Locally Matching the Noise Level of the Data," *Proc. Third Int. Conf. on Artificial Intelligence Applications on Wall Street*, New York, NY,1995, Addendum.

[12] Weiss S. and Kulikowski C., "Computer Systems That Learn," Morgan Kaufmann, 1991.

[13] Werbos P., "Beyond Regression: New Tools for Predicting and Analysis in the Behavioral Sciences," Harvard University, Ph.D. thesis, 1974. Reprinted by Willey & Sons, 1995.

[14] White H., "Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns," *Proc. IEEE Int. Conf. on Neural Networks*, Vol. 2, 1988, pp. 451-458.