Data Mining Using a Genetic Algorithm Trained Neural Network

Randall S. Sexton
Computer Information Systems
Southwest Missouri State University
901 South National
Springfield, Missouri 65804
Office: 417-836-6453
Fax: 417-836-6907
Email: RandySexton@smsu.edu
Web: http://www.faculty.smsu.edu/r/rss000f/

Randall S. Sexton is an Assistant Professor of Computer Information Systems at Southwest Missouri State University, Springfield, Missouri. He received his Ph.D. in Management Information Systems from the University of Mississippi. His research interests are in artificial intelligence, optimization, and neural networks. He has published work in a number of journals, including *European Journal of Operational Research*, *Decision Support Systems*, *OMEGA*, *End User Computing*, and *Journal of Computational Intelligence in Finance*.

Naheel A. Sikander
Computer Information Systems
Southwest Missouri State University
901 South National
Springfield, Missouri 65804
Email: nas705s@smsu.edu

Naheel A. Sikander is an undergraduate senior in Computer Information Systems at Southwest Missouri State University, Springfield, Missouri. She will be graduating in May, 2002 with honors and is currently applying for acceptance in a doctoral program in information systems. Her research interests are in neural networks, decision support systems, and artificial intelligence.

Data Mining Using a Genetic Algorithm Trained Neural Network

Abstract

Neural Networks have been shown to perform well for mapping unknown functions from historical data in many business areas, such as Accounting, Finance, and Management. Although there have been many successful applications of neural networks in business, additional information about the networks is still lacking, specifically, determination of inputs that are relevant to the neural network model. It is apparent that by knowing which inputs are actually contributing to model prediction a researcher has gained additional knowledge about the problem itself. This can lead to a parsimonious neural network architecture, better generalization for out-of-sample prediction, and probably the most important, a better understanding of the problem. It is shown in this paper that by using a modified genetic algorithm for neural network training, relevant inputs can be determined while simultaneously searching for a global solution.

KEY WORDS: Data Mining, Genetic Algorithm, Neural Networks, Artificial Intelligence, and Chaotic Time Series.

1. Introduction

The foremost reason why neural networks (NNs) are increasing in popularity is their ability to approximate unknown functions to any degree of desired accuracy as demonstrated by Funahashi (1989) and Hornik et al. (1989). In addition, NNs can do this without making any unnecessary assumptions about the distributions of the data. This makes it convenient for researchers, in that they can include any input variables that they feel could possibly contribute to the NN model. Although, it is likely that irrelevant variables are introduced to the model, the NN is expected to sufficiently learn to ignore these variables during the training process. It does this by finding weights associated to these irrelevant variables that when plugged into the NN would generate values that simply zero each other out, thereby, having no effect on the final output prediction. Although, this works fine for training data, when applied to observations that it has not seen (out-of-sample or testing data), these weights are going to generate values that are unlikely to zero each other out, causing additional error in the prediction. If, on the other hand, the research could identify the irrelevant variables, these variables could be excluded from the NN model and eliminate the possibility of introducing additional error when applied to out-of-sample data.

Although it is convenient for researchers to be able to include all available input variables into the model to extract a good solution, it also has the detrimental effect of making the NN a “black box” where they throw everything into the model but do not know why or how the network produces its output. Additional information about the problem can be obtained by identifying those inputs that are actually contributing to the prediction. For example, we could train a NN that predicted whether an employee would likely quit within the next year. As inputs, we could include employee information such as gender, age, length of employment, etc. A NN model that can accurately predict this outcome as well as identify the relevant inputs to the model, would be very beneficial in identifying reasons why employees were leaving. In order to test this algorithm we used the Henon Map (Schuster, 1995), a chaotic time series problem. Chaotic time series are interesting because they closely resemble real world financial data. As in financial data, without *a priori* knowledge of the solution, researchers would have difficulty in determining the number of lags to include in the network. By using the proposed algorithm to determine those variables that are relevant to prediction we have learned additional information about the problem.

Section 2 includes a background of literature on backpropagation and the genetic algorithm. Section 3 describes the GA method used in this study, which includes the base algorithm and the additional modification that we propose. Section 4 describes the problem and how the data was generated. Section 5 outlines how the GA determines the number of hidden nodes (architecture) and the training process. Section 6 reports the results of the experiment and the last section concludes with final remarks in section 7.

2. Background Literature

Since the majority of NN research is conducted using gradient search techniques, such as backpropagation, which require differentiability of the objective function, the ability for researcher to identify relevant variables, beyond trial-and-error, is eliminated. In this paper, a modified genetic algorithm is used for training a NN, which does not require differentiability of the objective function that will correctly distinguish relevant from irrelevant variables and simultaneously search for a global solution.

2.1. Backpropagation

Backpropagation (BP) is currently the most widely used search technique for training NNs. BP's original development is generally credited to Werbos (1993), Parker (1985) and LeCun (1986) and was popularized by Rumelhart et al. (1986a, 1986b). Although many limitations to this algorithm have been shown in past literature (Archer and Wang, 1993; Hsiung et al., 1990; Kawabata, 1991; Lenard et al., 1995; Madey and Denton, 1988; Masson and Wang, 1990; Rumelhart et al., 1986a; Subramanian and Hung, 1990; Vitthal et al., 1995; Wang, 1995; Watrous, 1987; White, 1987), its popularity continues because of many successes. An additional limitation to BP, that this paper is concerned with, is its inability to identify relevant variables in the NN model. This inability stems from the gradient nature of BP, which requires the objective function (usually the sum of squared errors (SSEs)) to be differentiable. This requirement prevents any attempt to identify weights in the models that are unnecessary, beyond pruning of the network. Pruning the network is simply eliminating connections that have basically no effect on the error term. This can be done by trial-and-error, saliency of weights, node pruning (Bishop, 1995). Also, there have been approaches to network construction, such as Cascade Correlation (Fahlman and Lebiere, 1990), which attempts to build parsimonious network architectures. A better approach might be to use an alternative algorithm, such as the GA, that is not dependent on derivatives to modify the objective function to penalize for unneeded weights in the solution. By doing so, the GA can search for an optimal solution that can identify those needed weights and corresponding relevant variables.

2.2. The Genetic Algorithm

The GA is a global search procedure that searches from one population of solutions to another, focusing on the area of the best solution so far, while continuously sampling the total parameter space. Unlike backpropagation, the GA starts at multiple random points (initial population) when searching for a solution. Each solution is then evaluated based on the objective function. Once this done, solutions are then selected for the second-generation based on how well they perform. Once the second generation is drawn, they are randomly paired and the crossover operation is performed. This operation keeps all the weights that were included in the previous generation but allows for them to be rearranged. This way, if the weights are good, they still exist in the population. The next operation is mutation, which can randomly replace any one of the weights in the population in order for a solution to escape local minima. Once this is complete, the generation is ready for evaluation and the process continues until the best solution is found. The GA works well searching globally because it searches from many points at once and is not hindered by only searching in a downhill fashion like gradient techniques.

Schaffer et al. (1992) found more than 250 references in literature for research pertaining to the combination of genetic algorithms and neural networks. In this past research, the GA has been used for finding optimal NN architectures and as an alternative to BP for training. This paper combines these two uses in order to simultaneously search for a global solution and a parsimonious NN architecture. Schaffer (1994) found that most of the past research using the GA as an alternative training algorithm has not been competitive with the best gradient learning methods. However, Sexton et al. (1998)

found that the problem with this past research is in the implementation of the GA and not its inability to perform the task. The majority of past implementations of the GA encode each candidate solution of weights into binary strings. This approach works well for optimization of problems with only a few variables but for neural networks with a large number of weights, binary encoding results in extremely long strings. Consequently, the patterns that are essential to the GA's effectiveness are virtually impossible to maintain with the standard GA operators such as crossover and mutation. It has been shown by Davis (1991) and Michalewicz (1992) that this type of encoding is not necessary or beneficial.

A more effective approach is to allow the GA to operate over real valued parameters (Sexton, 1998). The alternative approach described in Sexton et al. (1998) also successfully outperformed backpropagation on a variety of problems. This line of research in based on the algorithm developed by Dorsey & Mayer (1995) and Dorsey, Johnson and Mayer (1994).

The GA and its modifications used in this study follows in Section 3. Since the GA is not dependent on derivatives, a penalty value can be added to the objective function that allows us to search not only for the optimal solution, but one that also identifies relevant inputs to the model.

3. Method

The following is a general outline of how the GA searches for global solutions. A formal description of the algorithm can be found in Dorsey & Mayer (1995). The additional operations included in this study are also described.

3.1. Initialization

The GA randomly draws values to begin the search. However, unlike BP, which only draws weights for one solution, the GA will draw weights for a population of solutions. The population size for this study is set to 20 solutions, which is user defined and is based on past research (Dorsey & Mayer, 1995). Once the population of solutions is drawn, the training begins with this first generation.

3.2. Evaluation

Each of the 20 randomly drawn solutions in the population is evaluated based on a pre-selected objective function, which is not necessarily differentiable. Since our objective is to find a global solution that can identify relevant variables, an objective function is needed that will not only measure the error between estimates and real outputs, but will also include additional penalties for the number of non-zero connections in the model. The following objective function, Equation 1, takes the sum of squared errors and adds an additional penalty error for every connection (C), or weight, that is non-zero. The additional error is set to the root mean squared error (RMSE) or the typical error of one observation for the particular solution being evaluated. Consequently, a connection or weight is only eliminated if the SSE is increased by less than the RMSE. By setting the penalty equal to the RMSE, the NN is prevented from eliminating too many weights in the solution. Although, this objective function seems to work well for the problem in this study, the penalty value C assigned for non-zero connections is arbitrary. Additional research, beyond the scope of this study, is warranted for finding an optimal penalty assignment and is left for future research.

Equation 1

$$E = \sum_{i=1}^{N} \left(O_i - \hat{O}_i\right)^2 + C\sqrt{\frac{\sum_{i=1}^{N} \left(O_i - \hat{O}_i\right)^2}{N}}$$

Simplified, equation 1 is the Sum of Squared Errors (SSE) plus the product of C (number of nonzero weights for the solutions) and the Root Mean Squared Error (RMSE). One by one, each solution out of the population of 20 solutions is plugged into the network and a corresponding objective function value is determined, resulting in 20 objective function values. Based on these 20 values, a probability of being drawn in the next generation is assigned to each solution. For example, the best solution (lowest objective function value) out of the population is given the highest probability for being drawn for the next generation, while the worst solution (highest objective function value) is given the lowest probability. The first step in assigning probabilities for each solution is to find the distance away from the worst solution. For example, if we had errors of 50, 100, 200, and 500, we would take the absolute value of 50-500, 100-500, 200-500 and 500-500 resulting in 450, 400, 300, and 0. We would then take these distances and divide them by the sum of all the distances. This would give us probabilities of .4, .3, .2, .1, and 0. Therefore, when a new generation is drawn, the solution that had resulted in an error of 50 would have a 40% chance of being drawn, while the solution that had an error of 500, would have a 0% chance of being drawn. This completes the first generation.

3.3. Reproduction

The second generation is then drawn based on the assigned probabilities from the former generation. For example, the best solutions (ones with the smallest errors and therefore highest assigned probabilities) in the first generation are more likely to be drawn for the second generation of 20 solutions. This is known as reproduction, which parallels the process of natural selection or "survival of the fittest". The solutions that are most favorable in optimizing the objective function will reproduce and thrive in future generations, while poorer solutions die out.

3.4. Crossover

These 20 solutions, which only include solutions that existed in the prior generation, are now randomly paired into 10 sets of solutions. For each paired set of solutions, a random integer value in the range [1,*n*], with *n* being equal to the number of weights in a solution, is drawn to decide where the crossover operation is to take place. Once *n* is determined, all the weights above that value are switched between the pair resulting in two new solutions. For example, if a solution contains 10 weights, a random integer value is drawn from 1 to 10 for the first pair of solutions. For this example, we will assume the value selected is six. Every weight above the 6^{th} weight is now switched between the paired solutions, resulting in two new solutions for each pair. Once this is done for each pair of solutions, crossover is complete.

3.5. Mutation

To sample the entire parameter space, and not be limited only to those initially random drawn values from the first generation, mutation must occur. Each solution in this new generation now has a small probability that any of its weights may be replaced with a value uniformly selected from the parameter space.

3.6. Mutation2

In order for each solution to have the possibility of containing zero weights, an additional operation (mutation2) is conducted. Each solution in this new generation now has a small probability that any of its weights may be replaced with a hard zero. This is done to identify those weights in the solution that are not needed for estimating the underlying function.

Once reproduction, crossover, mutation, and mutation2 have occurred, the new generation can now be evaluated to determine the new probabilities for the next generation. This process continues until the maximum number of generations set by the user is reached.

4. Problem Data

Chaotic time series problems come from a class of problems called deterministic chaos. These types of functions are of great interest to researchers due to their similarity to chaotic behavior of economic and financial data. The specific chaos problem used for this study is known as the Henon Map (Schuster, 1995). Although the Henon Map equations seem to be simple, they produce apparently random, unpredictable behavior, which is very difficult to predict. Another characteristic of this chaotic system is its sensitivity to initial conditions. Starting from very close initial conditions, a chaotic system very rapidly moves to different states. This will allow us to test how well the NN picks up the underlying unknown function when tested on data derived from different initial conditions.

Henon Map

$$
\begin{aligned}
X_t &= 1 - 1.4X_{t-1}^2 + Z_{t-1} \\
Z_t &= 0.3X_{t-1} \\
X_0 &= 0.2, Z_0 = 0.2
\end{aligned}
$$

The training data was constructed by plugging the initial X and Z values at 0.2. Out of the 1,100 observations that were generated the first 1,000 observations were used for training and the last 100 were used for testing. Each observation, at this point, includes X_t and Z_t for inputs and X_{t+1}, and Z_{t+1} for outputs.

Since our main objective is to identify relevant inputs, we need to include irrelevant inputs into the data set so that we may test our algorithm. The values of X and Z are approximately in the ranges of [-1.28, 1.28] and [-.38, .38], respectively. In order to include irrelevant variables that were approximately equal to the relevant variables, we included 20 additional input variables drawn randomly from a uniform distribution from the range of X and 20 additional input variables from a uniform distribution from the range of Z. The data set now included 42 input variables and 2 outputs, where 40 of the inputs were random numbers. In order to demonstrate the robustness of the algorithm, 10 data sets were generated. Ten sets of 1,100 observations for the 40 irrelevant variable values were generated. Each set was then combined with the original 1,100 observations making 10 data sets that only differed in the values used for the irrelevant variables.

Since, chaos systems are very sensitive to the initial conditions, another test set of 1,000 observations was generated with the initial conditions for X_0 and Z_0 at 0.3. Again, for this test set, we combined these observations with 10 different sets of 40 irrelevant variables that were randomly drawn from a uniform distribution in the same ranges as the training data sets. In total, we have 10 training sets of 1,000 observations, 10 test sets of 100 observations, 10 additional test sets of 1,000 observations.

5. Genetic Algorithm Neural Network Architecture and Training

The GA, as presented by Dorsey et al. (1994), was used for this study with the inclusion of the Mutation2 operation and different objective function. The program was coded in Fortran for use on a personal computer. All runs were conducted on a 450-MHz

Pentium machine, using one-hidden layer and the standard sigmoid function. The only changes to the original algorithm were the addition of the mutation2 operation, modified objective function, and the method of determining the number of hidden nodes in the NN architecture.

The number of hidden nodes was automatically chosen in the following manner for each of the 10 different NNs. First, a network begins with one hidden node. The network trains for 100 generations, saving the objective function value as the BEST so far, and its corresponding weights and number of hidden nodes. An additional hidden node is then included into the network and trained for 100 generations. At the conclusion of this training, the new objective function value is compared with the BEST. If the new objective value is better than the BEST, it becomes the new BEST and the corresponding weights and number of hidden nodes are saved. This process continues until three consecutive hidden node additions do not find a better solution than the BEST. At which time, the number of hidden nodes and weights, corresponding with the BEST objective function value, are used for an additional 2,500 generations of training. Although, many more generations are possible, 2,600 generations was sufficient to show the GA's ability to find good solutions as well as the relevant input variables. The error measure used to evaluate how well the NNs performed is the RMSE.

6. Results for the Henon problem

Although, the main purpose of this paper is to test the GA's ability to identify relevant input variables in the data set, there still is a need to have a baseline comparison of the accuracy of the solutions. If the GA were able correctly to identify relevant variables but found solutions that were inferior to BP techniques the experiment would be much less meaningful. To do this, results from 50 BP networks were used to compare with the results in this study. Also, since the GA used in this study is constructive in nature when determining the number of hidden nodes in a network, another comparison was conducted with the Cascade-Correlation algorithm. A PC based NN software Neural Works Professional II/Plus by NeuralWare, was used for this comparison. The BP networks included the same data sets using 5 different network topologies: one-hidden layer networks with 2, 4, 8, 16, or 32 hidden nodes. Training was performed using the QuickProp algorithm, which is a variation of the backpropagation algorithm. For each of the 10 data sets, 5 different BP network topologies were used, totaling 50 different runs. A special note should be made about these comparisons; the results shown include the average errors for the GA trained networks (10 data sets), compared with BP's results out of 50 runs. In addition, the BP networks used the first test set of 100 observations for a validation set. A validation set allows the BP network to evaluate the current network solution by looking at the errors generated when applied to this validation test set. Although, the BP does not use these observations for training, the information from this validation set is used to determine termination of the network training. Once the errors for the validation set cease to decrease, even if further training would decrease the in-sample error, the network will terminate. This common practice is used to help deal with the over-training issue. The GA neural network did not use this test set in this manner. An additional 10 runs was conducted using the NeuralWare version of Cascade-Correlation (CC) algorithm in order to compare how well this algorithm produced parsimonious architectures. The default values and recommendations from the

NeuralWare were used in implementing CC. A comparison for the average In-Sample, Test1, and Test2 RMSE and standard deviations is made in Table 1.

(Place Table 1 approximately here)

As can be seen in Table 1, the GA found superior solutions over the BP networks. In fact, the worst GA solution out of the 10 trained networks was better than the best solution out of the 50 BP networks and 10 CC networks for in-sample and both test sets. During the course of training BP used the first test set (test1) in order to determine an appropriate termination of training. This is sometimes called a validation data set. Once the error for the test1 data set stops decreasing, the BP network terminates its learning. Although the BP network utilized this additional information in order to perform better for out-of-sample observations, the GA still outperformed the BP network for the test1 data set.

Since the BP algorithm is not capable of eliminating weights, it had to find weight combinations for those irrelevant variables that virtually zeroed each other out for the training data. As mentioned earlier, these weights might not zero out when applied to test sets, thereby introducing additional error to the out-of-sample estimates. This is demonstrated by the differences between in-sample error and test2 error, shown in Table 2 and again graphically in Graph 1. As can be seen in this table, the GA's average difference is much smaller than all 5 averages for BP networks, as well as CC network. It is interesting to note that as the BP networks increased in size, the differences between in-sample and test2 error also increased. As the number of hidden nodes increased, the number of weights connected to irrelevant variables would also increase. As the weights increased for these irrelevant variables, it makes intuitive sense that there would be more opportunity for out-of-sample estimates to contain additional error. Since the GA zeroed out unnecessary connections in the solution, when applied to out-of-sample data, the possibility of adding additional error was eliminated.

(Place Table 2 approximately here)

(Place Graph 1 approximately here)

In looking at the CC algorithm's ability to find parsimonious networks we found that out of the 10 runs it had an average of 5.5 hidden nodes per network. The GA not only found better solutions than CC, but did so with an average of 2.2 hidden nodes. Also, since CC does not eliminate individual weights, only hidden nodes and all the weights associated with them, it does not have the capability to discriminate between relevant and irrelevant input variables, which is the main purpose of this paper.
Now that the ability of the GA to find good solutions is established, a closer look can be made as to the GA's ability to identify relevant input variables in the data set. Out of the 42 inputs included in all 10 data sets, we know 40 of the input variables are completely irrelevant for finding the solution. By determining which inputs are contributing to the prediction, we have discovered additional knowledge about the problem.

The two relevant variables are numbers 41 and 42. For the 10 network solutions, the irrelevant variables (1-40) were correctly identified for all but 1 network. The fourth training set solution identified all but one of the irrelevant variables (variable 31) in the final solution. Since the networks were all terminated before fully converging, we thought it would be interesting to take this solution and train it an additional 2,400 generations to see if it could eventually identify this last irrelevant variable. With a total of 5,000 generations, this network correctly identified all 40 irrelevant variables and reduced the RMSE for in-sample, test1, and test2 to 1.12E-03, 1.07E-03, and 1.12E-03, respectively.

Based on reviewer comments another comparison was included that compared our algorithm to two other feature selection techniques. The first algorithm used was found in Neural Works Professional II/Plus by NeuralWare. By choosing the "prune" option this algorithm attempted to eliminate unnecessary weights in the architecture. In order to do this, we needed to select a fairly large network to train in order to prune the structure back. We decided that a network of 32 hidden nodes was sufficient for this task. It was found that for all 10 training sets that there was not one solution that eliminated all connections to an input variable. Also, the results were not significantly different from the original BP networks. A preprocessing method that attempted to reduce the number of input variables was also utilized for this comparison. Linear regression was conducted using STEPWISE in order to determine which variables were significant. Although this method identified the two relevant variables, it also included four irrelevant variables in the model. Taking these 6 variables we constructed new datasets and trained additional BP networks. It was found that these networks were not significantly different than the original BP networks. Also, this type of feature selection ignores possible interactions between variables.

7. Results for real world problems

In order to show how well our algorithm worked on real-world data we used four additional datasets from Prechelt (1994). The following briefly describes the 4 classification problems used for this study.

Card - The second data set is used to classify whether or not a bank (or similar institution) granted a credit card to a customer. This data set includes 51 inputs and 2 outputs. The exemplars are split with 345 for training, 173 for validation, and 172 for testing, totaling 690 exemplars. The inputs are a mix of continuous and integer variables, with 44% of the examples positive. This data set was created based on the "crx" data of the "Credit screening" problem data set from the UCI repository of machine learning databases.

Diabetes - The third data set is used for diagnosing diabetes among Pima Indians. This data set includes 8 inputs and 2 outputs. The exemplars are split with 384 for training, 192 for validation, and 192 for testing, totaling 768 exemplars. All inputs are continuous, and 65.1% of the examples are negative for diabetes. This data set was created from the "Pima Indians diabetes" problem data set from the UCI repository of machine learning databases.

Gene - This data set is used for detecting intron/exon boundaries in nucleotide sequences. The three classes of possible outputs included an intron/exon boundary (a donor), an exon/intron (an acceptor) boundary, or none of these. There are 120 inputs made up from 60 DNA sequence elements, which have a four-valued nominal attribute that was encoded to binary, using two binary inputs. The exemplars are split with 1588 for training, 794 for validation, 793 for testing, totaling 3175 exemplars. There are 25% donors and 25% acceptors in the data set. This data set "splice junction" was also created from the UCI repository of machine learning databases.

Horse - The horse data set is used for predicting the fate of a horse that has colic. The three classes include, will survive, will die, or will be euthanized. There are 58 inputs and 3 outputs to this data set. The exemplars are split with 182 for training, 91 for validation, and 91 for testing, totaling 364 exemplars. The distributions of classes are 62% survived, 24% died and 14% were euthanized. This data set was created from the "horse colic" problem database at the UCI repository of machine learning databases.

Here again we need to validate that our GA results are at least as good as BP networks so we trained each of these datasets with both algorithms. The error measure used for these problems was the percent classification error. As can be seen in Table 3, the GA outperforms BP.

(Place Table 3 approximately here)

In all four datasets the GA significantly reduced the number of input variables without the network losing its ability to classify. The reductions in input variables are shown in Table 4.

(Place Table 4 approximately here)

8. Conclusions and final remarks

In this study, the GA was found to be an appropriate alternative to BP for training neural networks that not only finds better solutions with a parsimonious structure, but can also identify relevant input variables in the data set. By using the GA in this manner, researchers can now determine those inputs that contribute to estimation of the underlying function. This can help with analysis of the problem, improved generalization, and network structure reduction. These results have demonstrated that a

NN can be more than just a “black box.” It was shown that a complex chaotic time series problem as well as real-world problems could be solved that outperformed traditional NN training techniques as well as discovering relevant input variables in the model. Based on these results, future research is warranted for additional experiments and comparisons using the GA for NN training.

References:

Archer, N. P., and Wang, S. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. Decision Sciences, 24, 1 (1993), 60-75.

Bishop, C. M., Neural Networks for Pattern Recognition, Clarendon Press, Oxford 1995.

Davis, L. (ed.). Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY (1991).

Dorsey, R. E.; Johnson, J. D.; and Mayer, W. J. A genetic algorithm for training feedforward neural networks. Advances in Artificial Intelligence in Economics, Finance and Management (J. D. Johnson and A. B. Whinston, eds., 93-111). 1, (1994), Greenwich, CT: JAI Press Inc.

Dorsey, R. E., and Mayer W. J. Genetic algorithms for estimation problems with multiple optima, non-differentiability, and other irregular features. Journal of Business and Economic Statistics, 13, 1 (1995), 53-66.

Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems*, Volume 2, 524-532. San Mateo, CA: Morgan Kaufamann.

Funahashi, K. I. On the approximate realization of continuous mappings by neural networks. Neural Networks, 2, 3 (1989), 183-192.

Hornik, K.; Stinchcombe, M.; and White, H. Multilayer feed-forward networks are universal approximators. Neural Networks, 2, 5 (1989), 359-366.

Hsiung, J. T.; Suewatanakul, W.; and Himmelblau, D. M. Should backpropagation be replaced by more effective optimization algorithms? Proceedings of the International Joint Conference on Neural Networks (IJCNN), 7 (1990), 353-356.

Kawabata, T. Generalization effects of k-neighbor interpolation training. Neural Computation, 3 (1991), 409-417.

Lenard, M.; Alam, P.; and Madey, G. The applications of neural networks and a qualitative response model to the auditor’s going concern uncertainty decision. Decision Sciences, 26, 2 (1995), 209-227.

LeCun, Y. Learning Processes in an asymmetric Threshold network. Disordered Systems and Biological Organizations, Springer Verlag, Berlin, (1986), 233-240.

Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin (1992).

Madey, G. R., and Denton, J. Credit evaluation with missing fields. Proceedings of the INNS, Boston, (1988), 456.

Masson, E., and Wang, Y. Introduction to computation and learning in artificial neural networks. European Journal of Operational Research, 47 (1990), 1-28.

Parker, D. Learning logic. Technical report TR-87. Cambridge, MA: Center for Computational Research in Economics and Management Science, MIT (1985).

Prechelt, L. PROBEN1 – A set of benchmarks and benchmarking rules for neural network training algorithms, Technical Report 21/94, Fakultat fur Informatik, Universitat Karlsruhe, Germany. Anonymous FTP:/pub/papers/techreorts/1994/1994-21.ps.gz on ftp.ira.uka.de, (1994)

Rumelhart, D. E.; Hinton, G. G.; and Williams, R. J. Learning internal representations by error propagation. Parallel distributed Processing: Exploration in the Microstructure of Cognition, Cambridge MA: MIT press, (1986a), 318-362.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. Learning representations by back propagating errors. Nature, 323 (1986b), 533-536.

Schaffer, J. D.; Whitley, D.; and Eshelman, L. J. Combinations of genetic algorithms and neural networks: A survey of the state of the art, COGANN-92 Combinations of Genetic Algorithms and Neural Networks, IEEE Computer Society Press, Los Alamitos, CA, (1992), 1-37.

Schaffer, J. D. Combinations of genetic algorithms with neural networks or fuzzy systems. Computational Intelligence: Imitating Life, J.M. Zurada, R.J. Marks, and C. J. Robinson, eds., IEEE Press, (1994), 371-382.

Schuster, H. (1995). Deterministic Chaos: An Introduction. Weinheim; New York: VCH.

Sexton, R. S.; Dorsey, R. E.; and Johnson, J. D. Toward a global optimum for neural networks: A comparison of the genetic algorithm and backpropagation. Decision Support Systems, 22, 2 (1998), 171-186.

Subramanian, V., Hung, M. S. A GRG-based system for training neural networks: Design and computational experience. ORSA Journal on computing, 5, 4 (1990), 386-394.

Vitthal, R.; Sunthar, P.; and Durgaprasada Rao Ch. The generalized proportional-integral-derivative (PID) gradient decent back propagation algorithm. Neural Networks, 8, 4 (1995), 563-569.

Wang, S. The unpredictability of standard back propagation neural networks in classification applications. Management Science, 41, 3 (1995), 555-559.

Watrous, R. L. Learning algorithms for connections and networks: Applied gradient methods of nonlinear optimization. Proceedings of the IEEE Conference on Neural Networks 2, San Diego, IEEE, (1987), 619-627.

Werbos, P. The roots of backpropagation: From ordered derivatives to neural networks and political forecasting. New York, NY: John Wiley and Sons, Inc., (1993).

White, H. Some asymptotic results for back-propagation. Proceedings of the IEEE Conference on Neural Networks 3, San Diego, IEEE, (1987), 261-266.

Table 1 – Average RMSE and Standard Deviations

	In-sample	Stdev	Test1	Stdev	Test2	Stdev
GA	5.37E-03	1.21E-03	4.93E-03	1.49E-03	5.40E-03	1.67E-03
CC	6.75E-01	2.82E-02	6.42E-01	3.45E-02	6.77E-01	2.83E-02
BP2	1.81E-01	1.70E-01	1.74E-01	1.68E-01	1.88E-01	1.73E-01
BP4	1.18E-01	4.07E-02	1.13E-01	3.97E-02	1.28E-01	4.29E-02
BP8	1.30E-01	1.80E-01	1.35E-01	1.73E-01	1.48E-01	1.83E-01
BP16	7.29E-02	3.18E-02	9.25E-02	3.32E-02	9.84E-02	3.45E-02
BP32	3.87E-02	1.22E-02	7.54E-02	1.76E-02	8.07E-02	1.92E-02

Table 2 – Differences between in-sample and test2 RMSE

GA	2.67E-05
CC	1.58E-03
BP2	7.84E-03
BP4	1.05E-02
BP8	1.85E-02
BP16	2.56E-02
BP32	4.20E-02

Graph 1

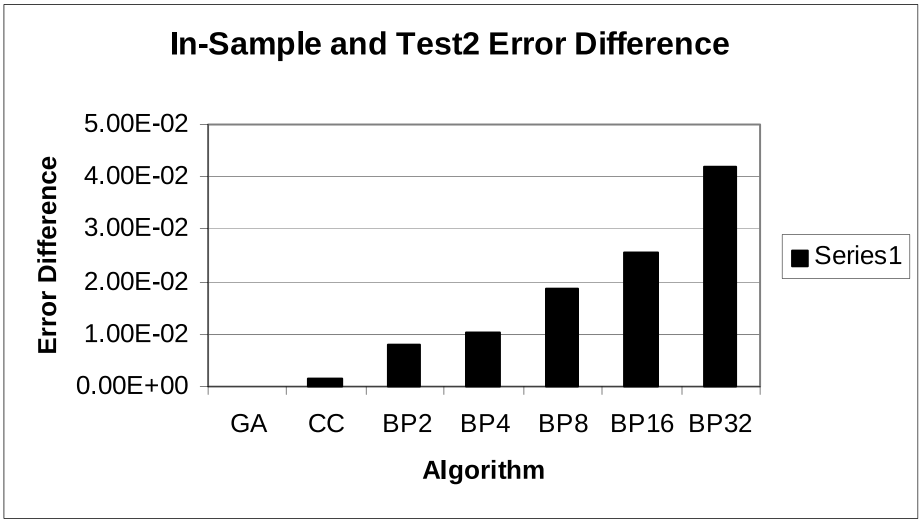
In-Sample and Test2 Error Difference
Error Difference
5.00E-02
4.00E-02
3.00E-02
2.00E-02
1.00E-02
0.00E+00
GA
CC
BP2
BP4
BP8
BP16
BP32
Algorithm
Series1

Table 3 – Classification Error

Problem	Classification error percentage	
	BP	GA
Card	15.1163	12.2093
Diabetes	27.6042	23.4375
Gene	25.3468	10.3405
Horse	25.2747	19.7802

Table 4 – Relevant Input Variable Reduction

Problem	Number of Inputs	Included Inputs	Percentage Reduction
Card	51	11	78.43%
Diabetes	8	5	37.50%
Gene	120	17	85.83%
Horse	58	13	77.59%