ELSEVIER

Decision Support Systems 22 (1998) 171–185

Decision Support Systems

Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation

Randall S. Sexton [a], Robert E. Dorsey [b,*], John D. Johnson [b]

[a] *Ball State University, Muncie, IN 47306, USA*
[b] *University of Mississippi, University, MS 38677, USA*

Abstract

The recent surge in activity of neural network research in business is not surprising since the underlying functions controlling business data are generally unknown and the neural network offers a tool that can approximate the unknown function to any degree of desired accuracy. The vast majority of these studies rely on a gradient algorithm, typically a variation of backpropagation, to obtain the parameters (weights) of the model. The well-known limitations of gradient search techniques applied to complex nonlinear optimization problems such as artificial neural networks have often resulted in inconsistent and unpredictable performance. Many researchers have attempted to address the problems associated with the training algorithm by imposing constraints on the search space or by restructuring the architecture of the neural network. In this paper we demonstrate that such constraints and restructuring are unnecessary if a sufficiently complex initial architecture and an appropriate global search algorithm is used. We further show that the genetic algorithm cannot only serve as a global search algorithm but by appropriately defining the objective function it can simultaneously achieve a parsimonious architecture. The value of using the genetic algorithm over backpropagation for neural network optimization is illustrated through a Monte Carlo study which compares each algorithm on in-sample, interpolation, and extrapolation data for seven test functions. © 1998 Elsevier Science B.V.

Keywords: Backpropagation; Delta-rule; Extrapolation; Genetic algorithm; Global search algorithms; Interpolation; Optimization

1. Introduction

Interest in business applications of artificial neural networks is growing rapidly. Evidence for this exists in both the academic and trade business literature. Increasingly, researchers are exploring artificial neural networks as a new tool for decision support. This is not surprising since the underlying functions controlling business data are generally unknown and the neural network offers a tool that can approximate not only the unknown function to any degree of desired accuracy as demonstrated by Funahashi [9] and Hornik et al. [11] but also its derivatives. [1] The vast majority of these studies rely on a gradient algorithm, typically a variation of backpropagation (BP), to obtain the parameters (weights) of the model. The well-known limitations of gradient search techniques applied to complex nonlinear optimization problems

[*] Corresponding author. Department of Economics, University of Mississippi, University, MS 38677, USA. Tel.: +1-601-232-7575.

[1] These studies indicate that one can compute *any* continuous function using linear summations and a single properly chosen nonlinear c.d.f. Technically, for a wide class of nonlinear functions NNs can provide arbitrary approximations to arbitrary functions in a variety of normed function spaces (e.g., functions in L_p spaces and functions in Sobolev spaces with a Sobolev norm) provided a sufficient number of hidden nodes (see Ref. [15]).

such as artificial neural networks have often resulted in inconsistent and unpredictable performance.

Many papers have attempted to address the problems associated with the training algorithm by imposing constraints on the search space or by restructuring the architecture of the neural network. A recent example of imposing constraints on the search space can be seen in Ref. [23] which proposes a 'fix' for certain classification problems by constraining the NN to only approximate monotonic functions. Restructuring the NN architecture requires the researcher to identify the correct number of hidden nodes to include in a NN. Although, trial-and-error is the most common method for determining this value, attempts to identify the optimum number is the subject of current research. One line of research explores Cascade Correlation, or Cascor [8], which builds a neural network topology while simultaneously identifying the appropriate connection weights. This technique has achieved encouraging success [13,1] with gradient search methods. It is not clear, however, whether any particular solution is chosen because it is the ideal parsimonious structure or because it is the easiest for gradient-based algorithms to optimize. Clearly this is an important direction for future research. Our research attempts to identify a robust optimization technique that will work regardless of the NN topology. In this paper we suggest that such constraints and restructuring may be unnecessary if a sufficiently complex initial architecture and an appropriate global search algorithm is used.

2. Background

Since its inception by Werbos [24], Parker [18], LeCun [14], and Rumelhart et al. [19,20], researchers have commonly trained neural networks with backpropagation (see Ref. [13]). The accuracy of any approximation depends on the selection of proper weights for the neural network, unfortunately backpropagation is a local search algorithm and thus tends to become trapped in local optima. As with any gradient search algorithm solutions are found, at least in part, on the serendipity of the initial random draw of weights. If these initial weights are located on local grades, the algorithm will likely become trapped at a local optimum. Even more sophisticated quasi-Newton methods such as Levenberg–Marquardt (see Ref. [2]) offer little improvement.

Researchers have used a variety of approaches to try and adjust for this characteristic of backpropagation. For example, parameters of the algorithm can be adjusted to affect the momentum of the search so that the search will break out of local optima and move toward the global solution. The correct value of these parameters, however, is not known a priori and is often problem-specific. Therefore, for any given problem, a wide variety of parameters must be tried to generate confidence that a global solution has been found. Another common method for finding the best (perhaps global) solution using BP is to restart the training at many random points. Again, the number of necessary starting points is unknown and generally varies significantly with the complexity of the problem.

A third type of adjustment is to restructure the NN architecture in such a way that the BP algorithm is more likely to obtain the global solution. Although there has been significant research in this area, see for example Ref. [16] or [22], there is no generally accepted heuristic for this approach and different researchers prefer different methodologies. One of the more promising directions has been to use genetic algorithms for exploring a variety of potential architectures for use with BP. The problem with this approach is that although, as the architecture becomes less complex the BP algorithm is more likely to be successful, the NN increasingly loses its ability to model complex relationships.

Finally, another approach often used is the one represented in Ref. [23], where significant constraints are applied to the permissible functional forms that the NN is permitted to approximate. Unfortunately, when a priori restrictions are placed on a model when the true model is unknown, the likelihood that the resulting model is the 'true' model may be reduced, (p. 24 of Ref. [4]). While these constraints can range from minimal to severe, it is often the case that the stronger the constraints the better the performance of the BP algorithm. In Ref. [4] the NN was severely constrained to only approximate monotonic functions. While it is unclear why the NN would be used at all if the function were known to be monotonic since there are far more efficient standard statistical methods for approximating such functions, we will demonstrate that none of these procedures are necessary if a global search algorithm is used.

The application of genetic algorithms to neural networks has followed two separate but related paths. First, genetic algorithms have been used to find the optimal network architectures for specific tasks. This is not the main direction of our research. For example, Todd [22] and Miller et al. [16] represented various network architectures as connection constraint matrices which were mapped directly into a bit-string genotype. Modified standard genetic operators were then used to act upon populations of these genotypes to produce successively higher fitness levels. While interesting, this line of work seems to sidestep the issue of a universal functional mapping since it leaves unresolved the question of whether the model's architecture perform's poorly on a given task, due to the appropriateness of the given architecture or rather the inability of the backpropagation learning rule to achieve a global solution on the given architecture. Another example of this problem may exist in the well-known study [20], which utilizes special architectures for solving the addition and negation problems. Both can be solved easily with generic structures using a genetic algorithm. These problems are difficult however, when using a generic structure and backpropagation. This may be why Rumelhart et al. [20] utilized their custom architectures.

The second direction involves optimization of the neural network using genetic algorithms for search. This is the direction of our research. However, most of this work (see for example Refs. [17,25,26]) uses a binary representation of the weights. In the work reported in this paper all weights are represented as real numbers in the genetic algorithm. [2]

In Section 3 we described the global search algorithm used for this study, the genetic algorithm. Also, we conducted a Monte Carlo comparison on seven test functions between the genetic algorithm and the backpropagation algorithm. We then examined a classification problem from Ref. [23] and showed that (a) the constraint of monotonicity may not be supported by the data, (b) a global search algorithm is able to achieve a superior classification function for a completely general NN and (c) the same algorithm can simultaneously find a parsimonious structure for the NN. In all of these comparisons our primary objective is to compare the accuracy of the models and no consideration is given to the speed of convergence. In general, backpropagation will converge to a solution much more quickly than a global search algorithm such as the genetic algorithm. Here, our focus is on the in sample and out of sample accuracy of the achieved solution.

3. The genetic algorithm

In two recent papers Dorsey and Mayer have demonstrated, first for limited problems [6] and then with more complex problems and an extensive Monte Carlo study [7] that the genetic algorithm (GA) performs exceptionally well at obtaining the global solution when optimizing difficult nonlinear functions. It was further demonstrated in Refs. [23,24] that the GA also works well when optimizing the NN, another complex nonlinear function. 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.

A formal description of the algorithm can be found in Ref. [7]. In general, the algorithm begins by randomly selecting a population of possible solutions. Each potential solution is a set of weights for the NN. This population is the first generation from which the genetic algorithm will begin its search for an optimal solution. The size of the population is set to 20, which is recommended in Ref. [7]. Thus, for a genetic algorithm trained NN, 20 sets of weights, are evaluated in each generation. Unlike BP which moves from one point to another, the genetic algorithm searches the weight space from one set of weights to another set, searching in many directions simultaneously. This enhances the probability of finding the global optimum.

[2] Binary representations have worked effectively in many applications. However, for the type of problems explored in this paper we have found that real number representations of the weights significantly reduce search time due to the reduced dimensionality of the search space.

The preselected objective function (not necessarily differentiable) is computed for each candidate solution. In this study we use the sum of squared errors for the objective function to be consistent with BP. A probability is assigned to each solution based on the value of the objective function. For example, using the sum of the squared errors, the solutions which result in the smallest sum of squared errors are assigned the highest probabilities. This completes the first generation. The second generation begins by randomly selecting a new population from the former. Twenty solutions are chosen with replacement so that good solutions are likely to be well represented in the new population and poor solutions are unlikely to be drawn. This is known as reproduction. The algorithm very generally parallels the process of natural selection hence its name. As in the saying 'survival of the fittest,' the traits most favorable in optimizing the objective function will reproduce and thrive in future generations, while weaker traits die out.

This new population of solutions (all of which existed in the prior generation) is next randomly grouped into pairs of solutions and a subset of the weights from each solution are switched with its paired solution (crossover). This creates two possible solutions, each with some parameters (weights) from each of the parent solutions. Finally, each solution has a small probability that any of its weights may be replaced with a value uniformly selected from the parameter space (mutation). This resulting set of solutions now becomes the new population or next generation, and the process repeats. This process continues until the initial population evolves to a generation that will best solve the optimization problem, ideally the global solution.

4. Monte Carlo study

In order to compare the effectiveness of the GA with commonly used versions of BP a Monte Carlo comparison was conducted on the following seven test problems. The first five constitute a variety of functions of two variables incorporating addition, multiplication, division and powers. The sixth problem is the well known Glass–Mackey chaotic time series and the last problem is a production function that has the ability to exhibit both increasing and diminishing marginal returns.

$$Y = X_1 + X_2 \tag{1}$$

$$Y = X_1 \cdot X_2 \tag{2}$$

$$Y = \frac{X_1}{|X_2| + 1} \tag{3}$$

$$Y = X_1^2 - X_2^3 \tag{4}$$

$$Y = X_1^3 - X_1^2 \tag{5}$$

$$Y_t = Y_{t-1} + 10.5\left[\frac{0.2Y_{t-5}}{1 + (Y_{t-5})^{10}} - 0.1Y_{t-1}\right] \tag{6}$$

$$Y = 5\arctan(x_1 - 50) + 10\arctan(x_2 - 100) + 15\arctan(x_3 - 150) \tag{7}$$

Fifty observations were used as the training set for the first five problems. The data for these problems was generated by randomly drawing the input variables uniformly from the arbitrary sets $X_1 \in [-100,100]$ and $X_2 \in [-10,10]$. For the sixth problem, 100 data points were generated from an initial point of [1.6,0,0,0,0]. For the seventh problem, 100 observations on the three independent variables were drawn uniformly from the range [0,200].

In order to test the networks, twenty additional data sets were constructed for the first five problems, consisting of 150 observations each. The first ten test sets for each problem were generated to test the ability of the optimized NN to interpolate. The interpolation test data was therefore also drawn from the sets $X_1 \in [-100,100]$ and $X_2 \in [-10,10]$, but did not include any common observations with the training set. The second ten test sets were generated to evaluate the ability of the optimized NN to extrapolate outside of the range of the training set. This data was drawn from $X_1 \in [-200,-101]$ and $X_1 \in [101,200]$ and for $X_2 \in [-20,-11]$ and $X_2 \in [11,20]$.

For the sixth problem ten interpolation data sets were generated by generating 150 data points from the randomly chosen starting points. Since the Glass–Mackey equation is stable there was no extrapolation data set. The ten interpolation data sets for the seventh problem each consisted of 150 data points drawn uniformly from [0,200]. The ten extrapolation data sets also contained 150 points but were drawn from the range [201,400].

Although the GA does not require normalization of the data, the BP programs all normalized the data. Therefore the training data was normalized from -1 to 1 for both algorithms, in order to have identical training and output data for comparison. The square root of the mean squared error, RMS, was the measure used for comparing the differences in the neural nets optimized with BP and the GA. Each network included six hidden nodes for all problems. There may well be better network architectures for the given problems, but since we are comparing the optimization methods of the network, we chose a common architecture for both. The first four problems all had two input nodes plus a bias term. The fifth problem had only one input node in addition to the bias term. The sixth problem used five input nodes for the data (five lagged values of the dependent variable) plus the bias term. Only two were necessary but this information is typically not known to the researcher. The last problem had three input nodes plus a bias term.

In estimating unknown functions, it is rare that a researcher can identify all contributing factors for any given output. These unidentified variables are then incorporated as noise or error in the model. In order to account for such real life situations, an extension to the above experiment was included that added an error term to the training output terms for two of the stated problems ($X_1 + X_2$ and $X_1 \cdot X_2$) and again estimated by the neural net with both algorithms. The error terms were randomly drawn from a normal distribution with a mean of zero and variance of 5 for $X_1 + X_2$ and a mean of zero and variance of 10 for $X_1 \cdot X_2$. The corresponding data sets for the problems with error were the same sizes and were drawn from the same ranges as those without error.

4.1. Training with backpropagation

Commercial neural network software as well as software downloaded from the Internet were preliminarily evaluated in order to determine which package would give the best estimates for the given problems. These included, Neural Works Professional II/Plus by NeuralWare®, Brain Maker by California Scientific, EXPO by Leading Markets Technologies and MATLAB by Math Works (using both the backpropagation and the Marquardt–Levenberg algorithms). Although the performance was similar for all programs, Neural Works Professional II/Plus by NeuralWare®, a PC-based neural network application seemed to give the best estimates and was chosen for the test series. In training each problem using BP, there were four factors that were manipulated in an effort to find the best configuration for optimizing each problem. They included the learning rate, momentum, test size, and the Logicon algorithm. The different combinations of the learning rate and momentum are introduced in order to try and find the right combination that will allow the solution to escape local minima but not skip over the global solution. The test size is defined as the number of observations in the training set that are passed through the network before evaluation and weight adjustments. For example, if the training set consisted of 50 observations and the test size was set to 10, then after 10 observations passed through the network the error for those 10 would be calculated and the weights adjusted accordingly before continuing on to the next 10 observations. Normally the delta rule provides adjustment after every training pair, but recent modifications of the algorithm allow for larger test sizes. The Logicon algorithm was introduced by Gregg Wilensky and Narbik Manukian in 1992 and was developed to achieve faster convergence by combining the advantages of closed and open boundary networks, into a single network [27]. Each of these factors was varied in an effort to reduce the chances of becoming trapped in local minima. While there are rules of thumb for setting these values, there is no set standard upon which a researcher can draw for deriving optimum configurations for training with backpropagation. Guidelines suggested by the Neural Works manual were used in selecting the values used.

4.2. Backpropagation training factors

We examined 16 different configurations of these parameters and the NN was trained on the data for each problem with each configuration. Ten replications from different starting values were performed for each configuration and in each case the NN was trained for 20,000 epochs, where an epoch is defined as one full pass through the training set. These combinations are shown in Table 1. Although, the learning rate and momentum values selected for this study seem relatively high they were systematically reduced by a constant value of 0.5 every 10,000 epochs, automatically by the BP software. This was done in order to decrease the likelihood of oscillation. Each replication was started with a new random seed. Thus, there were 160 training attempts for each problem. Out of this set of 160, the best weights (lowest sum of squared errors) and the corresponding configuration (parameter settings) were used as the starting point for additional training.

At that point the weights were saved and these weights were next used as the starting point for training an additional 100,000 [3] epochs up to a total of 1,000,000 epochs for each of the seven problems using the optimal parameter settings. Although, the training was still conducted on the original in-sample data sets, the criterion for stopping this training was the reduction of interpolation data error. During this last sequence of training the network, the interpolation error was checked every 10,000 epochs for a reduction. After 10 consecutive checks with no reduction in interpolation error the training halted. The set of weights that achieved the smallest interpolation errors was then saved for the comparison with the GA. This gives BP an added advantage over the GA since the GA training process did not include interpolation data.

This methodology differs from the standard methodology used to compare algorithms. Typically when two algorithms are compared each is run several times and the average performance is compared. However, backpropagation does not have a standard protocol. Numerous papers have been written suggesting one method over another. This leaves the researcher in the position of being uncertain about the optimal approach to take. We therefore tried 160 different combinations for each problem to obtain the best solution and then tried to improve that solution before any comparisons were made. Here our focus is on the best solution achieved by each algorithm rather than the average solution although we report the range of solutions in the tables for comparison. Table 2 provides the best, worst, mean and standard deviation across all different configurations used with the backpropagation algorithm and for the genetic algorithm.

Table 1
Alternate backpropagation training parameters

Logicon Algorithm	Test size	Learning rate			
		0.5		1	
		Momentum		Momentum	
		0.3	0.9	0.3	0.9
Off	1	10 rep	10 rep	10 rep	10 rep
	50	10 rep	10 rep	10 rep	10 rep
On	1	10 rep	10 rep	10 rep	10 rep
	50	10 rep	10 rep	10 rep	10 rep

Since the GA was implemented on a CRAY-YMP, precision and operation time could affect the comparison with the PC-based Neural Works, so a backpropagation algorithm written and optimized for a CRAY-YMP was acquired. [4] Using this program, the seven problems were again estimated with BP. Each problem was trained for 1,000,000 epochs 10 times each. Each replication was started with a different random seed. The best replication for each problem was then trained for 50,000,000 additional epochs. Although the CRAY-YMP version of BP was trained with many thousands more epochs than the PC version, it did not obtain any of the best solutions for BP. The NN that had the smallest error out of all 170 different replications for each problem across both platforms was then selected for comparison with the GA solution.

4.3. Training with the genetic algorithm

The genetic algorithm is a global search algorithm. Parameters of the algorithm were set to the

[3] Depending on where the error stopped decreasing, some problems were trained more than 100,000 epochs in order to reach 1,000,000. However, all were trained at least 100,000 beyond the last reduction in error.

[4] This software was provided by Roger W. Meier at the US Army Engineering Waterways Experimentation Station, Vicksburg, MS.

Table 2
Comparisons of in-sample RMS error for BP and GA trained NNs

Problem	BP: Best	GA: Best	BP: Worst	GA: Worst	BP: Mean	GA: Mean	BP: Standard deviation	GA: Standard deviation	*P* value
1	4.11E − 01	4.16E − 07	17.63	8.46E − 06	8.56	2.61E − 06	5.86	4.02E − 06	0.00
2	11.65	1.27E − 02	337.90	5.07E − 01	164.27	1.15E − 01	86.45	1.22E − 01	0.00
3	5.31	1.82E − 01	15.30	1.29	10.37	5.22E − 01	2.51	6.13E − 01	0.00
4	178.70	4.09E − 02	3253.78	2.20	1023.46	3.50 − 01	942.71	2.33	0.00
5	8459.91	3.06E − 02	201,063	1.09	102,707	2.56E − 01	55,719.38	5.60E − 01	0.00
6	1.33E − 01	2.53E − 02	5.19E − 01	1.29E − 01	3.83E − 01	7.91E − 02	1.05E − 01	5.34E − 03	0.00
7	4.79	3.47E − 01	13.50	2.49	10.56	1.32	2.50	2.29	0.00

values recommended by Dorsey and Mayer [7]. [5] The only parameters selected were the number of generations and the random seed. The genetic algorithm was run on each problem 10 times and trained in each case for 5000 generations. The replications differed only in the changing of the random seed. Although, the network could have trained further with more generations, it was not necessary to demonstrate the effectiveness of a global search algorithm. In each case the genetic algorithm had not converged but was stopped after the prespecified number of generations. If the GA were allowed to continue, it will converge arbitrarily close to the global solution but the additional search time was unnecessary for the comparison.

5. Results

In all seven problems the best test size for BP was one. Also, for these problems the Logicon algorithm generated sizeably inferior estimates. [6] The only factor that appeared to be problem specific was the learning rate. Table 3 shows the factors and levels that achieved the best results across the 160 different training runs for each problem.

The genetic algorithm was run 10 times each with different random seeds for the problems since there was no need to explore for an optimal search configuration. Results for the best, worst, mean and standard deviation for the in-sample RMS errors for both BP and the GA are given in the Table 2. Although there was not a lot of variation in the GA performance there was a significant amount in the BP solutions. This wide variability supports the need for the researcher to use many different configurations and a variety of starting values when using BP for optimizing a NN. In each case the error for the GA is different from zero since the GA was terminated after 5000 generations. Additional search time would permit convergence to a solution arbitrarily close to zero. This table also includes the RMS error means, standard deviations and *P*-values for all replications conducted for both BP and the GA. As can be seen in Table 2, the GA's in-sample results were significantly better than the BP in-sample results for all seven test functions. The total number of epochs and corresponding computational time is shown in Table 4.

The best GA and BP trained NN solutions for each problem were next compared on the ten data sets for each problem with respect to both interpolation and extrapolation. This comparison is shown in

Table 3
Optimal training parameters for backpropagation

Problem	Learning rate	Momentum	Test size	Logicon
1	1	0.9	1	Off
2	1	0.9	1	Off
3	1	0.9	1	Off
4	0.5	0.9	1	Off
5	1	0.9	1	Off
6	0.5	0.9	1	Off
7	1	0.9	1	Off

[5] Copies of the code for the genetic algorithm are available at 130.74.186.120/dorsey/abs/ga2.htm.

[6] Recommendations for much lower learning and momentum rates for the Logicon algorithm were tried, but there was no significant difference in effect.

Table 4
In-sample training comparisons

Problem	Number of epochs[a]		Execution time[b] (s)	
	BP	GA	BP	GA
1	1,000,000	100,000	1250	343
2	1,000,000	100,000	1250	343
3	1,000,000	100,000	1250	343
4	1,000,000	100,000	1250	343
5	1,000,000	100,000	1100	317
6	1,000,000	100,000	1350	508
7	1,000,000	100,000	1250	494

[a]One epoch is a complete pass through the data.
[b]Time in seconds based on execution time on a Pentium 83 MHz PC.
PC version for BP NeuralWorks Professional II/PLUS NeuralWare.

Table 6
Wilcoxon matched-pairs ranks test

Problem	Number of test sets out of 10 where the GA trained NN found superior solutions to the BP trained NN at the 0.99 level of significance	
	Interpolation	Extrapolation
1	10	10
2	10	10
3	10	10
4	10	10
5	10	10
6	10	N/A
7	10	10

Table 5. Even though the NN's trained with the GA had far fewer epochs, each of the best solutions for all seven problems was superior to the corresponding best BP solution. These results demonstrate the BP tendency to converge to a local solution.

A statistical comparison of in-sample, interpolation and extrapolation results was conducted using the Wilcoxon matched-pairs signed ranks (2-tailed *P* significance) test. This test incorporates information about the magnitude of the differences as well as the direction of the differences between the pairs being tested. The best estimates for both BP and the GA were used for the interpolation and extrapolation comparison using this test method. The routine from the SPSS for Windows software package was used. Table 6 provides a summary of these results. As can be seen, the solution obtained by the GA dominated the solution obtained by BP in every test set at the 99% level of significance.

Problem 4, $X_1^2 - X_2^3$, is used in Figs. 1 and 2 to graphically illustrate the function and the out-of-sample interpolation estimates from both the BP and GA methods. These two figures are constructed by including the first 50 out-of-sample interpolation estimates for each algorithm with their corresponding true values in an *XY* graph. The true values are shown on the horizontal axis and these values were also plotted on the graph in order to better show the actual difference in the algorithms estimates for these values. The forecast values from BP and the GA are plotted on the vertical axis in Figs. 1 and 2, respectively.

While it appears that both the BP and GA method have captured the trend of the function, the GA is obviously mapping more accurately over the training

Table 5
Out-of-sample comparisons RMS errors

Problem	Interpolation comparison				Extrapolation comparison			
	Mean RMS error for 10 data sets		Standard deviation for 10 data sets		Mean RMS error for 10 data sets		Standard deviation for 10 data sets	
	BP	GA	BP	GA	BP	GA	BP	GA
1	1.89	4.06E − 07	1.55	2.07E − 08	8.45	1.42E − 04	3.08	2.52E − 05
2	20.20	1.56E − 02	9.23	1.26E − 03	265.82	1.29	23.73	7.32E − 02
3	9.63	4.46E − 01	4.34	3.73E − 02	26.30	7.10	1.81E − 01	2.72E − 01
4	360.97	4.67E − 02	123.58	2.43E − 03	7527.68	10.47	895.07	1.18
5	25,692.83	2.10E − 02	7574.86	3.09E − 03	183,633.17	3681.31	17,278.26	416.69
6	1.22E − 01	3.36E − 02	1.38E − 02	1.37E − 03	N/A	N/A	N/A	N/A
7	5.95	1.95	5.97E − 01	3.87E − 01	31.77	1.03E − 01	1.66	3.16E − 03

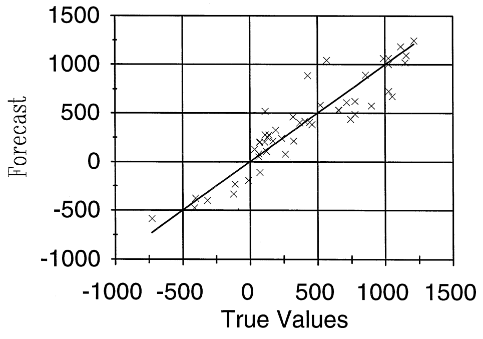


Fig. 1. BP interpolation forecasts. Line indicates problem-4 actual values.

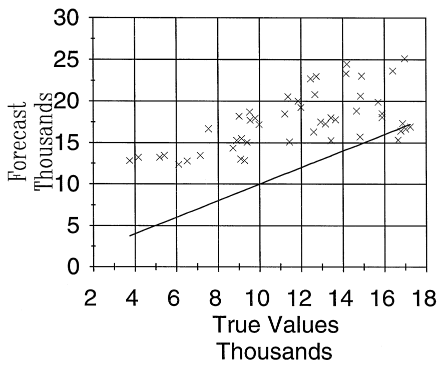


Fig. 3. BP extrapolation forecasts. Line indicates problem-4 actual values.

region. Further, if derivatives of the unknown function are important to the user, the GA clearly provides superior estimates of the true derivatives, while the BP method provides poor estimates of both slope and position. Figs. 3 and 4 show the extrapolation forecasts for the GA and BP trained NNs on the same problem.

A common concern of researchers using NNs is the issue of 'over fitting' or sometimes 'overtraining.' Various recommendations have been suggested in the literature to refrain from fully minimizing the objective function so that the NN will perform better out of sample. This is not an issue of 'overtraining' but rather one of over parameterization. If the objective function should not be fully optimized then clearly the researcher has the wrong objective function.

When there is no error in the data, as with the examples so far, then a NN cannot be overtrained. However, if the data contains errors and the NN is over parameterized there will be a tendency for the NN to attempt to fit each point including error and thereby not represent the true underlying function. We will discuss using the GA to reduce the parameterization of the model in Section 6. None the less, for a given level of parameterization a potential concern for researchers using the GA is how well a GA trained NN will perform outside the region of data used to train the NN (extrapolation) when errors exist in the data. It is important for the NN not to fit the data with errors but to generalize the pattern and approximate the true underlying function. To examine the relative performance of the two algorithms for this case, noise was added to the training data

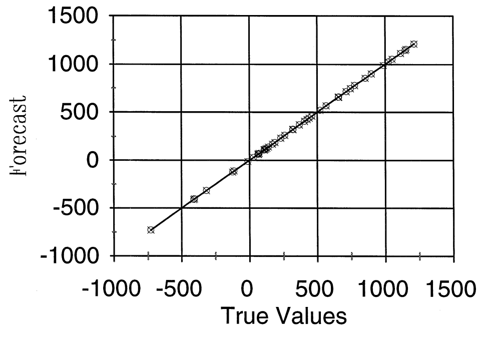


Fig. 2. GA interpolation forecasts. Line indicates problem-4 actual values.

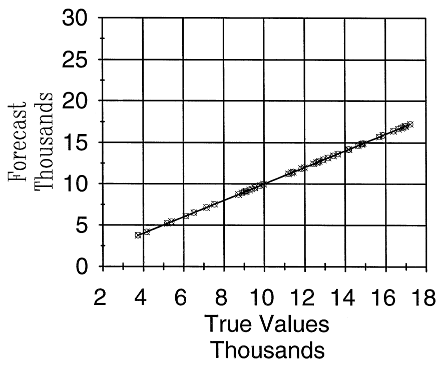


Fig. 4. GA extrapolation forecasts. Line indicates problem-4 actual values.

and the BP and GA were again used for training. The best training configuration for BP was used for 10 replications, changing the starting random seed in each case. The GA also was used to train the NN on the same data set for 10 replications. As can be seen in Table 7, all of the replications of the network trained with the GA had a significantly lower RMS error than its BP counterpart on the out-of-sample forecasts.

The data used for the out-of-sample evaluation was without error so that the NN could be evaluated with respect to the true functional form. After 20,000 epochs the best set of BP weights out of the ten replications was then used as the starting point for additional training to try and converge on a better solution. However, further training of this BP NN only increased the out-of-sample RMS error, while further training with the GA continued to decrease the RMS error. This pattern exhibited by the BP trained neural network is often referred to as 'over-training' but, in this case, is simply an indication of an insufficient search for an optimal solution. These results imply that the GA trained NN better approximates the true underlying function for the two data sets including error.

To illustrate the importance of finding global vs. local minima, one only has to compare the RMS errors of the two training methods, but in order to actually see why gradient methods are problematic, error surfaces are provided in Fig. 5. Each surface was constructed by using the best weights for the GA and BP for a particular problem. $X_1 \cdot X_2$ was chosen for this illustration because of its simplicity and nonlinearity. Two of the 16 possible input weights were chosen to vary by 0.01 increments at 50 points on each side of the original weight thus making a 101×101 grid. As each individual weight was varied, all other input layer weights were held constant. After each change, the output layer weights were reoptimized given the input weight change. After each weight change and reoptimization, a new SSE was calculated. This procedure generated a total of more than 10,000 SSEs which were used as points in order to plot the surfaces in Fig. 5. The plots show side by side comparisons of the GA and BP error surfaces. The weights chosen to vary for each plot were weight 1 of hidden node 1 and weight 2 of hidden node 2. There are many possible combinations of weights that could have been chosen for this illustration, and the others show similar results. As can be seen, in this region of the parameter space there are four local and one global solutions. The steep sides of the global valley are relatively close in proximity compared to the other four local valleys, so the probability of starting in an area leading to a local solution is much greater than the probability of

Table 7
RMS Comparison for functions including error

Parameters from training run	$X_1 + X_2 + \epsilon$[a]				$X_1 \cdot X_2 + \epsilon$[b]			
	Interpolation		Extrapolation		Interpolation		Extrapolation	
	BP	GA	BP	GA	BP	GA	BP	GA
1	4.14	1.27	36.65	3.47	16.02	1.66	1303.61	8.58
2	2.85	1.56	32.24	3.89	215.97	1.75	2037.61	8.97
3	2.82	1.82	34.28	5.12	49.27	1.67	1460.35	8.67
4	2.93	1.48	30.73	4.15	50.82	1.88	1399.33	8.84
5	6.95	1.57	34.27	9.38	30.56	1.64	1479.32	8.50
6	2.90	1.30	38.37	3.70	15.96	2.00	1321.45	9.52
7	2.80	2.03	34.59	5.87	19.73	2.13	1317.33	9.80
8	3.11	1.47	33.43	4.17	21.35	1.85	1322.68	9.79
9	2.99	1.50	34.33	3.50	29.86	1.75	1348.10	9.24
10	2.71	1.60	34.02	3.69	19.48	1.65	1307.22	8.48

[a]Error was drawn from a normal distribution [$\mu = 0$, $\sigma^2 = 5$].
[b]Error was drawn from a normal distribution [$\mu = 0$, $\sigma^2 = 10$].

starting the search in an area leading to a global solution. It is also apparent by this graph that if the search did start in a place leading to a local optimum, it would be difficult to obtain the global solution with a gradient search. Fig. 5 shows three additional magnifications of these points for comparison. Although the problem $X_1 \cdot X_2$ is relatively simple, the error surfaces are complex.

6. Classification

The classification example from Ref. [8] (pp. 21–23) discussed in Ref. [23] provides a further example of the flexibility of a global search algorithm. There are two independent variables in this data set corresponding to the use of two different computer operating systems. These are used to classify the users into two classes, nonmedical and medical users. Wang [23] assumes that the true underlying function partitioning the space is composed of two monotonic segments and that the connection point is known a priori. Under these assumptions, neural networks were constrained to approximate two monotonic functions which resulted in only four misclassifications. This was an improvement over the seven misclassifications reported in Ref. [10].

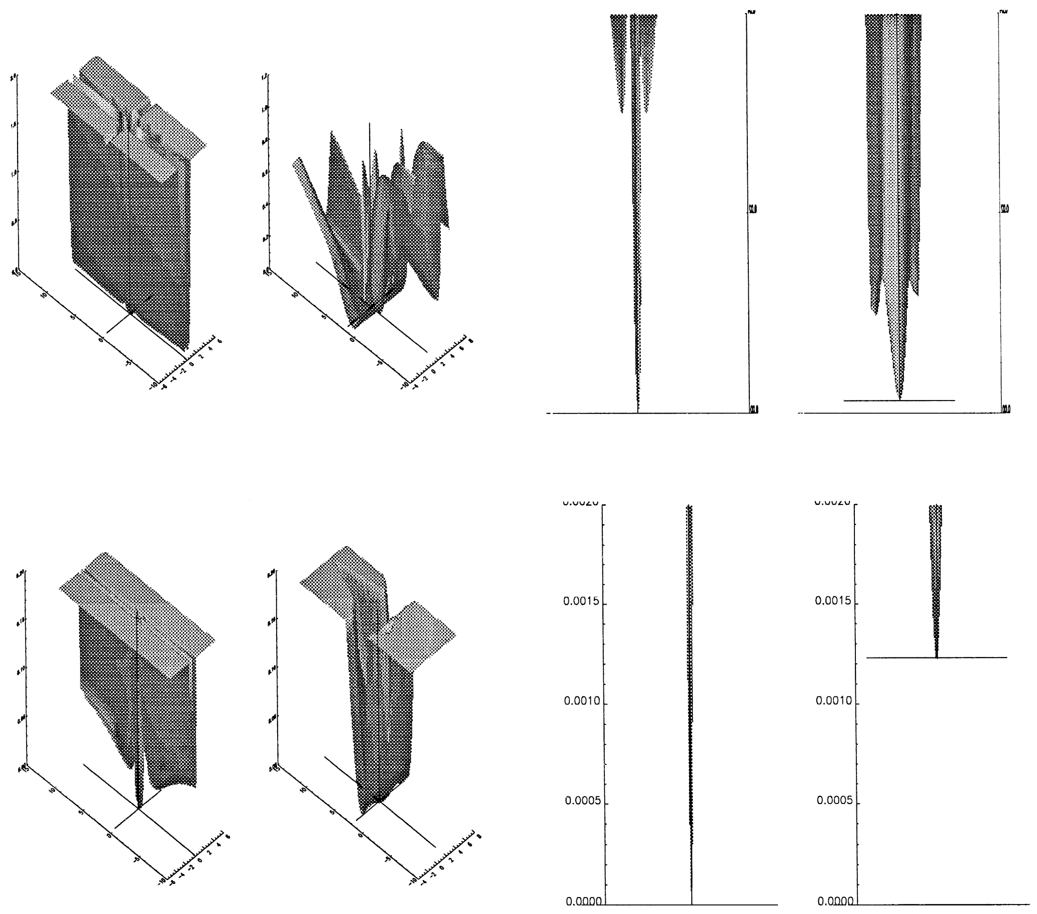
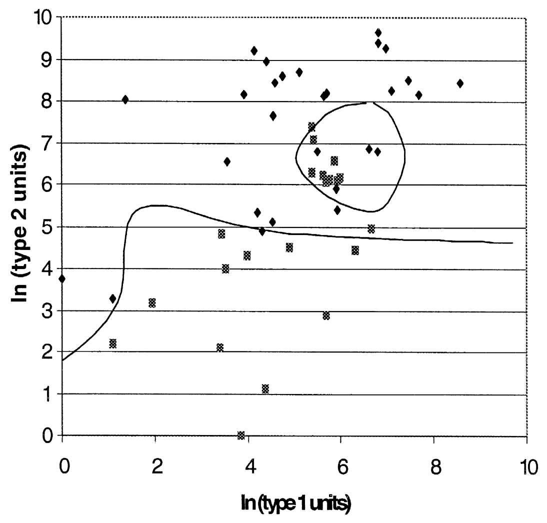
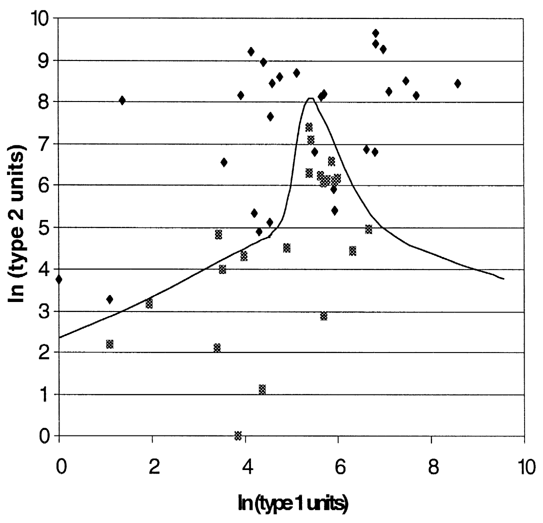


Fig. 5. Four levels of magnification of the GA and BP solutions to the $X_1 \cdot X_2$ problem. The vertical axis is the sum of squared errors. The horizontal axes correspond to two of the NN weights. The optimal values are centered on the axes and result in the smallest SSE. Four levels of magnification are shown. The least magnification is in the upper left, and then increasing to the lower left, upper right and finally lower right. In each pair the GA solution is on the left and the BP solution is on the right.

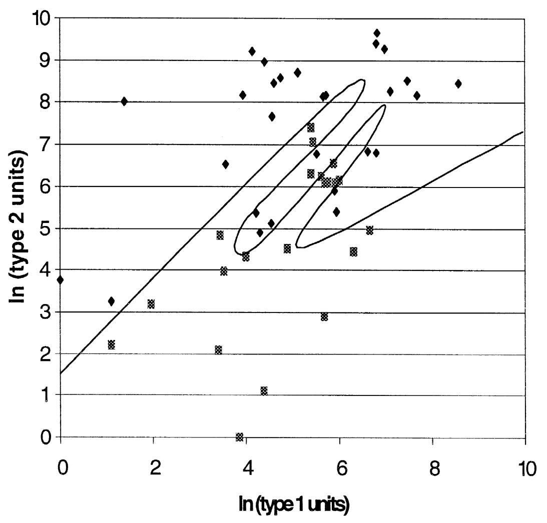
Most classification problems are far more complex than this example and it is typically not the case that the parameter space can be divided into a small number of monotonic segments. See for example Ref. [12] where the classification problem (financial impairment of insurance companies) was based on 11 financial measures. When the classification problem is more complex, the neural network can still serve as a valuable tool for separating the parameter space into the appropriate regions.



Non Parametric Classification from Hand(1981)



Monotonic NN from Wang (1995)



Unconstrained Neural Network

Fig. 6. Partitions of classification problem.

To demonstrate the ability of a feedforward network optimized with a global search algorithm we trained a neural network using the data from Ref. [26]. The architecture of the neural network consisted of two input nodes, six hidden nodes and one output node. The neural network trained with the genetic algorithm reduced the number of misclassifications from the four reported in Ref. [23] to 1. Without a priori knowledge of the true functional relationship this would appear to be a superior solution to the ones reported in Refs. [10,23]. Fig. 6 illustrates the classification data and the GA solution.

The issue again arises however, of complexity of the solution and the possibility of over parameterization of the neural network model. While it may be more accurate for the training sample if it is over parameterized it may perform poorly out of sample. This can be addressed by modifying the objective function for the neural network from the traditional minimization of the sum of squared errors to an objective function of the form:

$$\min\left\{ \beta M + \Sigma |Y_i - \hat{Y}_i| \right\}.$$

Here, M is the number of nonzero connection weights and β is the relative weight allocated to minimizing complexity relative to accuracy. The sum of the absolute errors is arbitrary and could as well be the sum of squared errors. When this objective function was used on the above problem with a value of β set at 10, the neural network again achieved an accuracy of only one error but at the same time eliminated the eight connection weights corresponding to two of the hidden nodes. Thus solving the problem with the same accuracy but with a neural network of only four hidden nodes. If prior knowledge of the problem implied that this solution was still too complex, then by simply increasing the value for β and reoptimizing, a less accurate but less complex solution can be achieved. For example, to obtain monotonic functions similar to those presented in Ref. [23] for the problem in Ref. [26], two NNs with six hidden nodes each were trained to replicate the reported monotonic functions. The genetic algorithm was then allowed to eliminate unnecessary connections and found that eight connections could be dropped and still maintain the same accuracy.

7. Conclusions

Artificial neural networks offer researchers a highly versatile new tool for flexible form estimation. Unfortunately, the successful utilization of this procedure requires that the ANN be fully optimized. Many researchers are currently using local search algorithms such as backpropagation for optimization and are thus likely to obtain local optima instead of the global solution. We demonstrate that such local solutions often perform poorly on even simple problems when forecasting out of sample.

Researchers have attempted to address this problem with ad hoc procedures such as stopping optimization at suboptimal solutions (not over training) or adjusting the neural network architecture to make optimization easier. We demonstrate that if a global optimization algorithm is used these arbitrary procedures are not necessary. Further we show that by using a global search algorithm such as the genetic algorithm, the objective function can be set to balance the trade-off between the over parameterization of the model that may over fit the data and a parsimonious ANN that can provide a more robust solution.

Acknowledgements

Drs. Johnson and Dorsey are supported in part by the Mississippi Alabama Sea Grant Consortium.

References

[1] P.K. Coats, F.L. Fant, Recognizing financial distress patterns using a neural network tool, Financial Manage. 22 (3) (1993) 142–156.

[2] M. Demuth, M. Beale, Neural Network Toolbox, Users' Guide, The Math Works, Natick, MA, 1994.

[3] R.E. Dorsey, J.D. Johnson, W.J. Mayer, The Genetic Adaptive Neural Network Training (GANNT) for generic feedforward artificial neural systems, School of Business Administration Working paper, 1992. Available at http://www.bus.olemiss.edu/johnson/ compress/compute.htm.

[4] R.E. Dorsey, J.D. Johnson, W.J. Mayer, A genetic algorithm for the training of feedforward neural networks, in: J.D. Johnson, A.B. Whinston (Eds.), Advances in Artificial Intel-

ligence in Economics, Finance, and Management, Vol. 1, JAI Press, Greenwich, CT, 1994, pp. 93–111.

[5] R.E. Dorsey, J.D. Johnson, M.V. Van Boening, The use of artificial neural networks for estimation of decision surfaces in first price sealed bib auctions, in: W.W. Cooper, A.B. Whinston (Eds.), New Direction in Computational Economics, Kluwer Academic Publishers, The Netherlands, 1994, pp. 19–40.

[6] R.E. Dorsey, W.J. Mayer, Optimization using genetic algorithms, in: J.D. Johnson, A.B. Whinston (Eds.), Advances in Artificial Intelligence in Economics, Finance, and Management, Vol. 1, JAI Press, Greenwich, CT, 1994, pp. 69–91.

[7] R.E. Dorsey, W.J. Mayer, Genetic algorithms for estimation problems with multiple optima, non-differentiability, and other irregular features, J. Bus. Econ. Stat. 13 (1) (1995) 53–66.

[8] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, Advances in Neural Information Processing Systems, Vol. II, Morgan Kaufmann, San Mateo, CA, 1990, pp. 524–532.

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

[10] D.J. Hand, Discrimination and Classification, Wiley, Chichester, 1981.

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

[12] C.S. Huang, R.E. Dorsey, M.A. Boose, Life insurer financial distress prediction: A neural network model, J. Insur. Regulat. 13 (2) (1995) 131–167.

[13] R.C. Lacher, P.K. Coats, S.C. Sharma, F.L. And Fant, A neural network for classifying the financial health of]a firm, Eur. J. Operation. Res. 85 (1) (1995) 53–66.

[14] Y. LeCun, Learning processes in an asymmetric threshold network, Disordered Systems and Biological Organization, Springer Verlag, Berlin, 1986, pp. 233–240.

[15] T.-H. Lee, H. White, C.W.J. Granger, Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests, J. Economet. 56 (1) (1993) 269–290.

[16] G.F. Miller, P.M. Todd, S.U. Hedge, Designing Neural Networks Using Genetic Algorithms, Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1989.

[17] D.J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, BBN Systems and Technologies Technical Report, 1989.

[18] D. Parker, Learning logic, Technical Report TR-87, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, 1985.

[19] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by backpropagating errors, Nature 323 (1986) 533–536.

[20] D.E. Rumelhart, G.G. Hinton, R.J. Williams, Learning internal representations by error propagation, Parallel Distributed Processing: Exploration in the Microstructure of Cognition, MIT Press, Cambridge, MA, 1986, pp. 318–362.

[21] L.M. Salchenberger, E.M. Cinar, N.A. Lash, Neural networks: A new tool for predicting thrift failures, Decision Sci. 23 (1992) 899–916.

[22] P.M. Todd, Evolutionary methods for connectionists architectures, Psychology Dept., Stanford University, unpublished manuscript, 1988.

[23] S. Wang, The unpredictability of standard backpropagation neural networks in classification applications, Manage. Sci. 41 (3) (1995) 555–559.

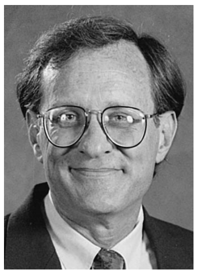
[24] P. Werbos, Beyond regression: New tool for prediction and analysis in the behavioral sciences, unpublished PhD thesis, Harvard Univ., 1974.

[25] D. Whitley, T. Hansen, The GENITOR algorithm: Using genetic recombination to optimize neural networks, Working paper, Computer Science Department, Colorado State Univ., 1989.

[26] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: Optimizing connections and connectivity, Parallel Comput. 14 (1990) 347–361.

[27] G. Wilensky, N. Manukian, The projection neural network, International Joint Conference Neural Networks, Vol. II, 1992, pp. 358–367.

Randall Sexton is an Assistant Professor of Management at Ball State University. He received his PhD in Management Information Systems at the University of Mississippi. His research interests include algorithm development and neural networks.



Robert Dorsey is an Associate Professor of Economics and Finance at the University of Mississippi. He received his PhD in Economics and Finance at the University of Arizona. Prior to attending graduate school he worked for fifteen years in the private sector. His research interests include computational methods, experimental economics and artificial intelligence. His Web page is at: http://www.bus.olemiss.edu/dorsey dorsey.htm.

John D. Johnson is an Associate Professor of Management Information Systems at the University of Mississippi. He received his PhD in Economics from Texas A&M University in 1987. He has worked primarily in nonlinear estimation focusing on the genetic algorithm for optimization of neural networks for flexible form approximation. His current work also deals with intelligent agents, data mining, data warehousing, the Internet and parallel processing. You can find him on the Internet at: http://www.bus.olemiss.edu/johnson/share/johnson.htm.