A Genetic Cascade-Correlation Learning Algorithm*

Mitchell A. Potter

Computer Science Department
George Mason University
Fairfax, VA 22030 USA
e-mail: mpotter@cs.gmu.edu

Abstract

Gradient descent techniques such as back propagation have been used effectively to train neural network connection weights; however, in some applications gradient information may not be available. Biologically inspired genetic algorithms provide an alternative. Unfortunately, early attempts to use genetic algorithms to train connection weights demonstrated that exchanging genetic material between two parents with the crossover operator often leads to low performance children. This occurs because the genetic material is removed from the context in which it was useful due to incompatible feature-detector mappings onto hidden units. This paper explores an approach in which a traditional genetic algorithm using standard two-point crossover and mutation is applied within the Cascade-Correlation learning architecture to train neural network connection weights. In the Cascade-Correlation architecture the hidden unit feature detector mapping is static; therefore, the possibility of the crossover operator shifting genetic material out of its useful context is reduced.

1 Problem Statement

Although gradient descent techniques such as back propagation have been used effectively to train feedforward neural network connection weights, researchers have experimented with evolving an optimal set of connection weights with biologically inspired genetic algorithms. Genetic algorithms are adaptive search algorithms in which a population of individuals representing alternative solutions to a problem are allowed to propagate based on a measure of their fitness (Holland 1975). As the population evolves, the fitness of the individuals increases through the application of genetic recombination operators. Genetic algorithms have several advantages over the back propagation algorithm. First, because genetic algorithms work on a population of solutions in parallel there is less chance of converging to

*Published in Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 123-133, IEEE Computer Society Press, 1992

a suboptimal local minimum solution. Second, the node transfer functions in a network trained with back propagation must be differentiable (Rumelhart et al. 1986). This is not a restriction in the genetic algorithm approach because the derivative of the transfer function is not needed in the computation of a gradient. Third, back propagation is generally used with feedforward networks containing no loops. Modifications to the algorithm are required to support recurrent networks (Pineda 1987). Genetic algorithms place no restrictions on network topology because they require no backward propagation of an error signal.

The efforts to train neural networks with genetic algorithms have achieved mixed success. Montana and Davis have trained a neural network using a genetic algorithm to do line characterization in the domain of sonar data analysis (Montana and Davis 1989). They have reported encouraging results in which their genetic algorithm outperformed back propagation. In addition, Whitley and Hanson have used genetic algorithms to train neural networks to solve some of the classic feedforward network test cases; such as the exclusive-or problem, 424-encoder, and two-bit adder (Whitley and Hanson 1989). However, the Whitley group found that when traditional genetic algorithms were applied to larger networks they often failed to find acceptable connection weight solutions (Whitley et al. 1989, 1990, and 1991).

The approach these researchers have taken is to create a population of individuals whose chromosomes represent alternative connection weight assignments for the network. The connection weights may be encoded in the chromosomes simply as a string of floating point numbers or with a more traditional binary encoding in which each real-valued connection weight is mapped to a segment of a string of bits. The population is initialized with individuals consisting of random connection weight values. The genetic algorithm is then substituted for back propagation to adjust the connection weights by recombining individuals in the population until a member evolves to an acceptable level of fitness. The fitness of an individual is determined by computing the sum squared error as training patterns are fed forward through the network; therefore, the genetic algorithm acts as a minimization function.

Traditionally genetic algorithms have relied heavily on crossover, a genetic recombination operator in which genetic material from the chromosomes of two parents is exchanged to create offspring. However, crossover has been of limited use in evolving connection weights in large networks. The Whitley group has hypothesized that this is because parent networks may contain incompatible feature-detector mappings onto hidden units. Therefore, exchanging genetic material between two parents often leads to low performance children because the genetic material is removed from the context in which it was useful. They achieved better neural network connection weight optimization results by emphasizing mutation, a genetic operator in which alleles are randomly modified. In a neural network training environment this consists of mutating one or more connection weight values to create a child.

Montana and Davis took an alternative approach in achieving their successful results on the sonar array problem by creating a new set of genetic operators specifically tailored to the domain of feedforward neural network connection weight optimization. By designing domain specific operators they were able to achieve an effective blend of exploration through mutation and exploitation through recombination while avoiding the problems associated with the crossover operator.

This paper explores a third approach in which a traditional genetic algorithm using standard two-point crossover and mutation is applied within the Cascade-Correlation learning architecture developed by Fahlman to learn neural network connection weights (Fahlman 1990). In the Cascade-Correlation architecture both neural network topology and connection weights are evolved simultaneously. Furthermore, the network is evolved one hidden unit at a time. Once a hidden unit is added, its input weights do not change for the remainder of the training; i.e., the unit becomes a permanent feature-detector. This reduces the possibility of the crossover operator shifting genetic material out of its useful context.

2 Technical Approach

2.1 Cascade-Correlation

In the Cascade-Correlation algorithm the network begins as a two-layer feedforward architecture in which each input unit has a weighted connection to each output unit. This two-layer network is trained until learning approaches an asymptote. Hidden units are then added one at a time through the application of two separate learning cycles. The Quickprop algorithm, a heuristic second-order gradient descent technique, is used for both of these cycles (Fahlman 1988).

The first cycle begins by creating a single new hidden unit and giving it a weighted connection from each input unit. The input weights for the new hidden unit are then trained by adjusting them to maximize the equation

$$Covariance = \sum_p \left| \sum_o (V_p - \overline{V})(E_{p,o} - \overline{E_o}) \right| \tag{1}$$

where o is a network output unit index, p is a training pattern index, V is the output of the hidden unit, and E is the network sum squared error; i.e., the covariance is maximized between the output of the hidden unit and the error from the network. Training patterns will be fed forward through the network until the covariance approaches an asymptote.

In the second learning cycle the output of the new hidden unit is given a weighted connection to each output unit. The entire set of connections to the output units (from all hidden and input units) will then be trained by minimizing the sum squared error equation

$$Error = \sum_p \sum_o (D_{p,o} - Y_{p,o})^2 \tag{2}$$

where o is a network output unit index, p is a training pattern index, D is the desired output, and Y is the actual output. This cycle also continues until learning approaches an asymptote.

If the error does not drop below a threshold value another hidden unit is created and the learning cycles repeat. As new hidden units are added the input connection weights of the previously installed hidden units remain fixed; that is, the units become permanent feature detectors. Later hidden units have input connections from all previously installed hidden units along with connections from all input units. Figure 1 shows a Cascade-Correlation network consisting of two inputs, a fixed bias signal, and two outputs, after the addition of three hidden units. The black connection boxes represent hidden unit input weights that will remain fixed throughout the rest of the training process while the white connection boxes represent output unit weights that will continue to be adjusted.

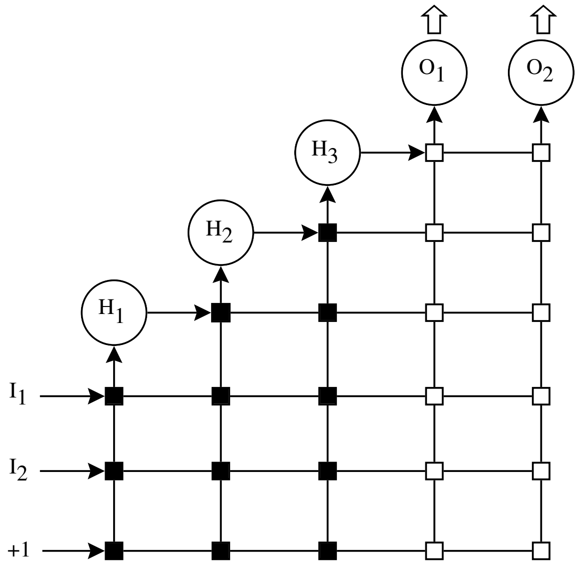

Figure 1: Cascade-Correlation Network After the Addition of Three Hidden Units

2.2 Genetic Algorithm

In the Genetic Cascade-Correlation algorithm, Quickprop is replaced with a genetic algorithm in both learning cycles. The advantages of using a genetic algorithm over the Quickprop algorithm are similar to those mentioned earlier with respect to back propagation. Specifically, they consist of reducing the possibility of convergence to suboptimal local minimum solutions and support for applications in which gradient information is not available. Although Fahlman addresses the potential problem of gradient descent convergence to suboptimal local minimum solutions in the correlation phase of Cascade-Correlation by training a population of "candidate units" in parallel and choosing the most highly correlated one as the hidden unit to be added, he uses a small population of four to eight individuals and does not allow them to interact with one another. Genetic algorithms typically maintain much larger populations and exploit the interaction of individuals through the crossover recombination operator.

The genetic algorithm used within the two learning cycles is based on the generational model in which a single generation consists of:

Selecting a set of individuals from the population to be reproduced using a proportional non-deterministic selection strategy

Recombining the individuals using two-point crossover and mutation

Evaluating the new individuals and inserting them back into the population with random replacement.

Each learning cycle continues to evolve its population until 100 generations have elapsed or until learning approaches an asymptote.

A separate population of fifty individuals is maintained for each of the two learning cycles. An individual chromosome consists of floating point alleles representing real-valued connection weights. Each individual in the population of the hidden unit input connection cycle consists of connection weights from each input unit, from the fixed bias, and from each previously installed hidden unit. Output connection cycle individuals each consist of the entire set of output connection weights. Because the number of connection weights in both cycles increases as hidden units are added to the network, the genetic algorithm supports a variable chromosome length.

At the beginning of each hidden unit input connection cycle, individuals are initialized with random connection weights in the range $(-1.0, 1.0)$. The first output unit cycle also initializes its population with individuals consisting of random connection weights in the range $(-1.0, 1.0)$. However, all output connection weights continue to be adjusted throughout the training process; therefore, information acquired during early cycles needs to be transferred to later cycles. The initialization approach taken for later output unit cycles that achieves this transfer yet maintains sufficient population diversity is to add to copies of the five best individuals from the previous output unit cycle new connection weights in the range $(0, -C)$, where C is the covariance achieved in training the hidden unit input connections. This range was chosen as a result of the heuristic that a unit whose output is positively correlated with an error from another unit can best reduce that error with a negatively weighted connection. Similarly, if the correlation is negative a positive connection weight will best reduce the error.

Selection consists of stochastic sampling with replacement. Sampling probabilities are proportional to scaled fitness. When new individuals are added to the population, old individuals are randomly selected for replacement. The best individual from each generation is preserved into the next generation. The hidden unit input connection cycle uses the correlation of the hidden unit output with the sum squared network error as a measure of fitness. The output unit cycle uses the sum squared network error as a measure of fitness.

The genetic recombination operators are entirely generic and consist of standard two-point crossover with a 0.6 probability of application to each chromosome and an adjustably disruptive mutation with a 0.2 probability of application to each floating point allele. The crossover operator cuts chromosomes on connection weight boundaries. The mutation operator includes a "temperature" parameter τ and consists of the addition of a random delta distributed uniformly in the range $-\tau$ to $+\tau$ to a connection weight. A larger value of τ results in more disruptive mutations. A fixed τ value of 0.5 was used in the output connection weight cycle. The τ value for the hidden unit input connection cycle is more application dependent and fixed values in the range 1.0 to 4.0 were used. Appropriate values of τ were determined empirically.

Although the genetic algorithm was implemented specifically for this project, partially due to the requirement that it be embedded within the Cascade-Correlation architecture, the algorithm itself is not restricted to the domain of training neural network connection weights.

3 Empirical Results

An implementation of the Genetic Cascade-Correlation algorithm is evaluated by comparing its performance to a Quickprop Cascade-Correlation program written by Fahlman. The programs differ only in the method used for training the connection weights. The two benchmarks used for this comparison are the two-spiral problem and the N-bit parity problem; both used by Fahlman in a comparison of Quickprop Cascade-Correlation to back propagation (Fahlman 1990).

The primary metric used to compare the programs is the number of epochs required to train the network. This is illustrated in the left graph of each pair below by plotting the sum squared network error as a function of the number of training epochs. In interpreting these graphs, note that a Genetic Cascade-Correlation epoch consists of a single forward-pass of all training patterns through the network and a population evolution of one generation. A Quickprop Cascade-Correlation epoch consists of a single forward-pass of all training patterns through the network, the computation of gradient information, and the use of this gradient information to adjust the connection weights. The number of epochs was chosen as a metric over computation time to reduce the significance of implementation details. In the current implementation a Quickprop epoch requires approximately one order of magnitude less computation than a genetic epoch.

A secondary metric is the number of hidden units required. This is illustrated below by the right graph of each pair showing the maximum covariance achieved as hidden units are added to the network.

All comparisons were run ten times and the results averaged to produce the graphs shown below. Every experiment converged to a solution. The Genetic Cascade-Correlation experiments produced normal distributions while the Quickprop experiments produced extremely heavy tailed distributions. None of the distributions were significantly skewed. Welch's two-tailed t-test was used to check the comparisons for statistical significance. (Welch 1947).

3.1 Two-spiral Problem

The two-spiral problem has two inputs representing an (x, y) coordinate and one output that is $+0.5$ for half of the training set and -0.5 for the other half. The coordinates for the $+0.5$ and -0.5 outputs form two interlocking spirals. The problem is to classify each input coordinate as a $+0.5$ coordinate or a -0.5 coordinate; that is, to determine in which of the two spirals the coordinate belongs. The training set consists of 194 input/output pairs forming interlocking spirals that loop around the origin three times. This problem is very difficult for back propagation networks to solve.

The graphs shown in Figure 2 compare the genetic and Quickprop performance on the two-spiral problem. The results were generated using sigmoid transfer functions for hidden units and output units. The Quickprop implementation maintained a population of eight candidate units. As described earlier the genetic implementation maintained two populations of fifty individuals (one for each of the two learning cycles), and used a crossover rate of 0.6, a mutation rate of 0.2, a hidden unit input connection cycle mutation temperature of 4.0, and an output unit cycle mutation temperature of 0.5. The genetic algorithm solved the two-spiral problem in an average of 1395 epochs and required an average of 33 hidden units while the Quickprop algorithm solved the two-spiral problem in an average of 1815 epochs and required an average of 17 hidden units. Welch's t-test produced a statistically significant P value of less than 0.001 for both the primary and secondary metric comparisons. The ten genetic experiments had a standard deviation of 154.9 epochs and 4.2 hidden units while the Quickprop experiments had a standard deviation of 142.0 epochs and 1.6 hidden units. The

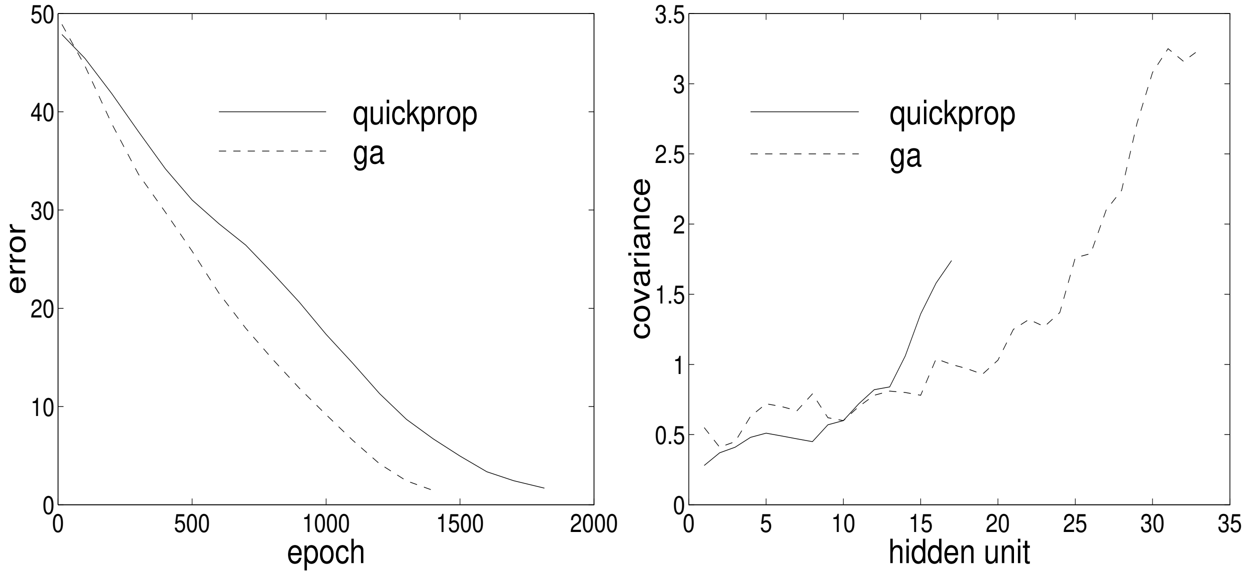

Figure 2: Comparison of Genetic and Quickprop Implementation of Two-spiral Problem

fastest convergence of the genetic algorithm was in 1141 epochs and the slowest convergence was in 1630 epochs. The fastest convergence of the Quickprop algorithm was in 1639 epochs and the slowest convergence was in 1981 epochs. The smallest genetically evolved network contained 28 hidden units and the largest contained 42 hidden units. The smallest network trained by Quickprop contained 15 hidden units and the largest contained 20 hidden units.

By increasing the number of generations each hidden unit is evolved by adjusting the parameters controlling the detection of asymptotic convergence; a solution containing only 13 hidden units was found with the genetic algorithm. However, over 4000 epochs were required.

In addition, the two-spiral problem was used to determine the effectiveness of the genetic crossover operator. Figure 3 compares the performance of the genetic implementation with and without this operator. All other parameters have the same values as previously described. When the crossover operator is deactivated, the average number of epochs required to solve the problem increased to 1533 and the average number of hidden units increased to 36. The experiments had a standard deviation of 227.5 epochs and 5.8 hidden units. The fastest convergence was in 1212 epochs and the slowest convergence was in 1904 epochs. The smallest network contained 28 hidden units and the largest contained 47 hidden units. The results indicate that the crossover operator may be making a positive contribution to the solution of the problem. When the operator is active, solutions are found in fewer epochs with a smaller number of hidden units. However, Welch's t-test produced a statistically insignificant P value of 0.13 for the primary metric comparison and 0.18 for the secondary metric comparison.

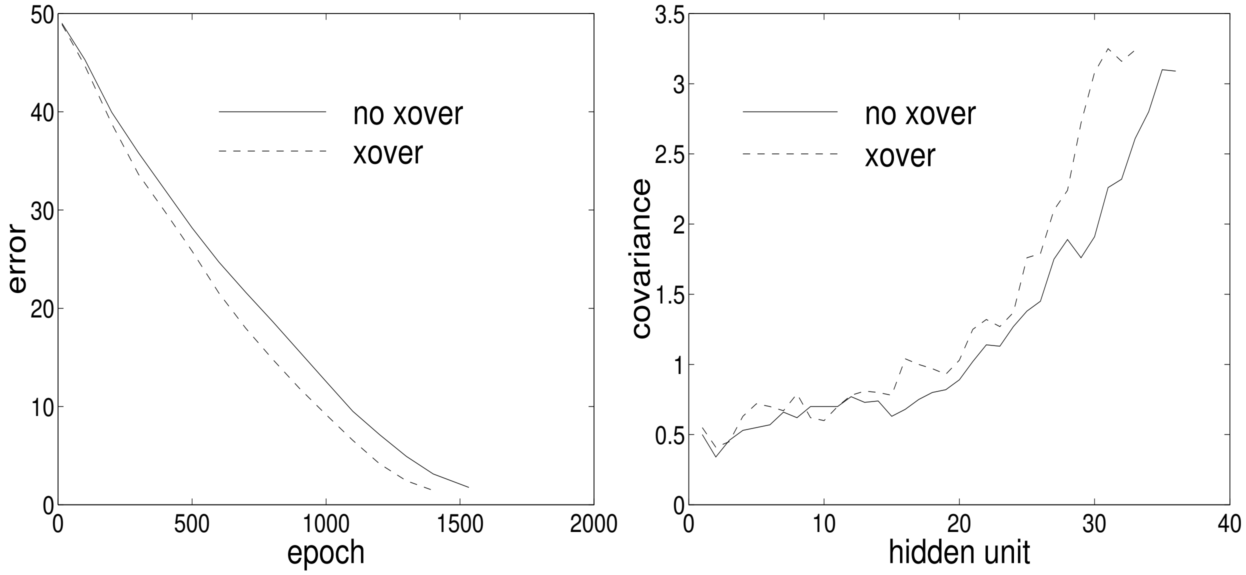

Figure 3: Comparison of Genetic Implementation of Two-spiral Problem With and Without the Crossover Operator

3.2 N-bit Parity Problem

The N-bit parity problem has N inputs (an N value of two is equivalent to the XOR problem) and a single output unit representing whether there is an even or odd number of one-bits in the input. An output of $+0.5$ represents odd parity and an output of -0.5 represents even parity. The training set for this problem consists of all 2^N possible training pairs.

The graphs shown in Figure 4 compare the genetic and Quickprop performance for the eight-input case. The results were generated using Gaussian transfer functions for hidden units and sigmoid transfer functions for output units. All other parameters had the same values as described earlier with the exception of the mutation temperature used in the genetic hidden unit input connection cycle. The best results were achieved with this mutation temperature set to a value of 1.0. The genetic algorithm solved the parity problem in an average of 629 epochs and required an average of 17 hidden units while the Quickprop algorithm solved the parity problem in an average of 569 epochs and required an average of 6 hidden units. Welch's t-test produced a statistically insignificant P value of 0.16 for the primary metric comparison and a statistically significant P value of less than 0.001 for the secondary metric comparison. The ten genetic experiments had a standard deviation of 85.2 epochs and 3.3 hidden units while the Quickprop experiments had a standard deviation of 96.9 epochs and 0.9 hidden units. The fastest convergence of the genetic algorithm was in 535 epochs and the slowest convergence was in 787 epochs. The fastest convergence of the Quickprop algorithm was in 446 epochs and the slowest convergence was in 702 epochs. The smallest genetically evolved network contained 12 hidden units and the largest contained 22 hidden units. The smallest network trained by Quickprop contained 4 hidden units and the largest contained 7 hidden units.

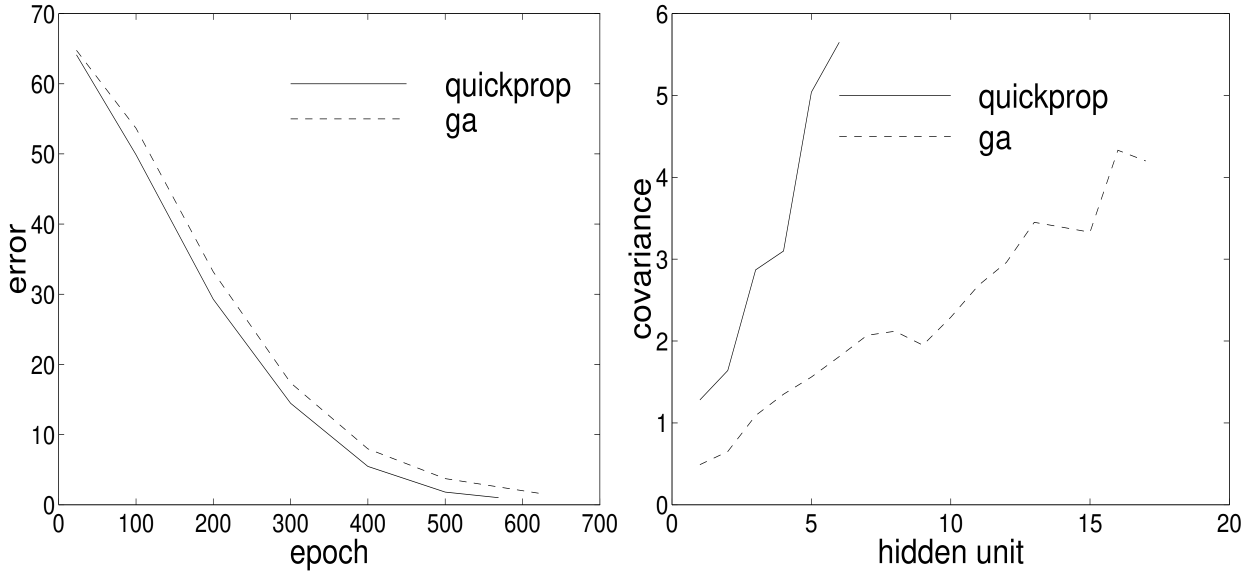

Figure 4: Comparison of Genetic and Quickprop Implementation of 8-bit Parity Problem

4 Conclusions

In conclusion, a genetic algorithm has been applied within the Cascade-Correlation architecture to train connection weights. The performance of the Genetic Cascade-Correlation program when applied to the two-spiral and eight-bit parity problems compares favorably with the original Cascade-Correlation program that used the Quickprop algorithm to train connection weights. Although Quickprop solved both problems with fewer hidden units, the genetic algorithm required fewer epochs in the case of the two-spiral problem and only slightly more epochs than Quickprop in the case of the eight-bit parity problem. If the creation of compact networks is desirable, the genetic algorithm can be adjusted to find solutions with fewer hidden units at the expense of additional training epochs. The performance of the genetic algorithm is encouraging considering it uses no gradient information (thus does not require differentiable transfer functions) and depends entirely on generic recombination operators. The advantage of the Quickprop algorithm is that it is significantly more efficient than the genetic algorithm when gradient information is available. However, in applications such as non-linear neural control where an error gradient is sometimes not computable, genetic algorithms may provide a reasonable alternative. Future work will consist of applying the Genetic Cascade-Correlation algorithm to such an application.

A comparison of the genetic algorithm performance with and without the crossover operator did not demonstrate a statistically significant increase in performance when standard two-point crossover was applied. However, earlier attempts to use genetic algorithms to train neural networks demonstrated the crossover operator significantly decreased the performance due to incompatible feature-detector mappings onto hidden units. The empirical results of this paper indicate that the application of crossover within the Cascade-Correlation architecture does not suffer from this defect. Future work may lead to alternate genetic representations within the Cascade-Correlation architecture that will enable the crossover operator to demonstrate a significant performance contribution.

The results are inconclusive as to whether the genetic algorithm reduces the possibility of convergence to suboptimal local minimum solutions. Both Genetic Cascade-Correlation and Quickprop Cascade-Correlation consistently found acceptable solutions to the two-spiral and eight-bit parity problems.

Acknowledgements

The author thanks Scott Fahlman and Carnegie Mellon University for making available the source code of the Quickprop Cascade-Correlation simulator.

References

[1] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, 1988.

[2] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[3] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[4] D.J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, 1989.

[5] F.J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[6] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Volume I*, pages 318–362. MIT Press, 1986.

[7] B.L. Welch. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34:28–35, 1947.

[8] D. Whitley, S. Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 562–569. Morgan Kaufmann, 1991.

[9] D. Whitley and T. Hanson. Optimizing neural networks using faster more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396. Morgan Kaufmann, 1989.

[10] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.