Breeding Swarms: A New Approach to Recurrent Neural Network Training

Matthew Settles
sett4263@uidaho.edu

Paul Nathan
nath5573@uidaho.edu

Terence Soule
tsoule@uidaho.edu

Department of Computer Science
University of Idaho
Moscow, ID 83843

ABSTRACT

This paper shows that a novel hybrid algorithm, Breeding Swarms, performs equal to, or better than, Genetic Algorithms and Particle Swarm Optimizers when training recurrent neural networks. The algorithm was found to be robust and scale well to very large networks, ultimately outperforming Genetic Algorithms and Particle Swarm Optimization in 79 of 80 tested networks. This research shows that the Breeding Swarm algorithm is a viable option when choosing an algorithm to train recurrent neural networks.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods; F.1.1 [**Models of Compuation**]: Self-modifying machines—*Recurrent Neural Networks*

General Terms

Algorithms

Keywords

Breeding Swarms, Particle Swarm Optimization, Genetic Algorithms, Recurrent Neural Networks

1. INTRODUCTION

Recurrent Neural Networks (RNN) are networks of neurons with feedback connections and are computationally more powerful than standard feedforward neural networks. In principle, RNN can implement almost any arbitrary sequential behavior. Today, recurrent neural networks are being used in a wide variety of applications including time-series prediction, speech recognition, music composition [5, 8, 15]. However, training RNN to perform a certain task is known to be a very difficult problem. Currently no ideal training algorithm exists.

In this paper we present a novel hybrid algorithm, Breeding Swarms (BS), to train RNN. The Breeding Swarms algorithm combines elements of both Genetic Algorithms (GA) and Particle Swarm Optimizers (PSO) to form a more robust algorithm. The goal is to train a strongly connected recurrent network that produces periodic behavior similar to that seen in biological neurons. The algorithm is tested on a variety of different network topologies, ranging from small networks with 8 neural connections, to large networks with 6723 neural connection.

The next section discusses previous research and gives an overview of the standard GA and PSO models. Section 3 presents the Breeding Swarm model developed for this research. Section 4 describes the RNN model used, the test problem and test parameters. The final two section discuss the results of the experiments and presents some conclusions.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

2. BACKGROUND

RNNs are NNs in which nodes in the 'later' layers of the network may feed back to nodes in 'earlier' layers. RNNs have several significant features missing from feedforward NNs. Feedforward NNs are limited to input vectors of fixed dimension; data received in a more general format may need extensive preprocessing [9]. A typical example of this difference occurs in time series data. A feedforward NN is limited to looking at a window of data of fixed size, determined by the number of input nodes. Associations in the data that extend beyond this window cannot be found by the NN. In contrast, a RNN's recurrent connections allow the network to 'store' information received earlier by feeding it back to an earlier layer of the network. Thus, the RNN's calculations are not limited by its input window.

Traditional neural network (NN) training algorithms, such as back propagation, are based on gradient descent. They systematically and incrementally reduce the output error. Although gradient descent approaches are very effective for a wide range of problems, they are generally restricted to finding local minimum and may get stuck in flat regions of a search space. In such cases the typical recourse is to restart the algorithm from a new random location. In contrast population based searches are not restricted to a local search and can easily move across flat regions. However, they tend to be slower and so are only appropriate for particularly difficult problems.

Unfortunately, RNNs are very difficult to train using gradient descent based approaches. First, the additional connections greatly expand the overall search space making local minima and flat regions much more likely. Second, gra-

dient decent training requires 'unfolding' the network. To train a network in which a particular input influences the network (through the recurrent connections) N time steps into the future requires training the network as if it were a feedforward network with N times as many layers [9]. This greatly increases the time required for training and, more importantly, the difficulty of training the RNN.

Population based approaches do not suffer these drawbacks. Because they are performance (fitness) based no unfolding is required. Furthermore they are much better suited to the complex search spaces created by RNNs. There have been numerous successful examples of using evolutionary approaches to train RNNs [2,7,11,14,16]. However, most have either evolved the topology and weights or used a hybrid algorithm that evolved the topology and used a local search or gradient descent search for the weights.

In previous work, a GA and PSO were tested and compared when evolving the weights for a fixed topology RNN [17]. Both the GA and PSO were successful in evolving RNNs to produce a periodic output in response to a fixed input signal.

In this research we chose to use fixed topologies and only evolve the weights of the RNN. A fixed topology makes comparisons to Genetic Algorithms and Particle Swarm Optimizers simpler. In addition fixed topologies also allows us to use a more typical representation and standard operators.

2.1 Genetic Algorithms

Genetic algorithms were first introduced by Holland in the early 1970's [10] and have been widely successful in optimization problems.

A real-valued generational Genetic Algorithm(GA) is used in these experiments. The GA uses a chromosome consisting of real values. The two best individuals are copied into the next generation (elitism). Tournament selection is used, with a tournament of size 2. The initial weights were randomly chosen in the range $[-0.5, 0.5]$. Gaussian mutation was used, with mean 0.0 and variance reduced linearly each generation from 0.7 to 0.4. Each weight in the chromosome has probability of mutation 0.1.

The crossover operator chosen for use in this experiment was blended crossover (BLX-α) [6]. In blended crossover, two parents are selected using some selection scheme. Each gene in the offspring is then calculated by randomly choosing a position in the range $[min_i - \Delta \cdot \alpha : max_i + \Delta \cdot \alpha]$. Where $min_i = min(x_i, y_i)$, $max_i = max(x_i, y_i)$ and $\Delta = |x_i - y_i|$. In this experiment α was chosen to be 0.1. The crossover rate is 0.6.

2.2 Particle Swarm Optimizer

One of the first implementation of particle swarm optimizers(PSO) was to train neural networks. As described by Eberhart and Kennedy, the PSO algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals (refereed to as particles) adapts by returning stochastically toward previously successful regions in the search space and is influenced by the successes of their previous best results and the global best result [13].

A standard particle swarm optimizer was used to train the recurrent neural network. The PSO consists of many particles/individuals, where each particle keeps track of its position, velocity, best position thus far, and current fitness. The position and velocity vectors refer to the particle's position and velocity within the search space, they are real valued vectors, with one value for each network weight. Each particle also keeps track of its current fitness (analogous to population members in GA), which is obtained by evaluating an error function at the particle's current position. The best fitness value thus far is retained as well as the particles position at that fitness value.

During each generation every particle is accelerated toward the particles previous best position and the global best position. This is achieved by calculating a new velocity term for each particle based on its current velocity, the distance from its previous best position, and the distance form the global best position. An inertia weight, reduced linearly each generation, is multiplied by the current velocity and the other two components are weighted randomly to produce the new velocity value for this particle, this in turn affects the next position of the particle during the next generation. Thus, the governing equations are:

$$\begin{aligned} \vec{v_i}(t) &= w \cdot \vec{v_i}(t-1) + c_1\varphi_1(\vec{p_i} - \vec{x_i}(t-1)) \\ &\quad + c_2\varphi_2(\vec{p_g} - \vec{x_i}(t-1)) \quad (1) \\ \vec{x_i}(t) &= \vec{x_i}(t-1) + \vec{v_i}(t) \quad (2) \end{aligned}$$

Where, $\vec{x_i}$ is particle i's position vector, $\vec{v_i}$ is particle i's velocity vector, $\vec{p_i}$ is particle i's previous best position vector and $\vec{p_g}$ is the global best particle's position vector. The parameter w is the inertia weight. Variables c_1,c_2, φ_1 and φ_2 are social parameters and random numbers in the range [0.0,1.0], respectively.

3. THE BREEDING SWARM MODEL

Both Angeline and Eberhart have suggested that a hybrid combination of the GA and PSO models could produce a very effective search strategy [1,4]. Our goal is to introduce an adjustable hybrid GA/PSO model. Our results show that with the correct combination of GA and PSO, the hybrid can outperform, or perform as well as, both the standard PSO and GA models.

The hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection, crossover and mutation from GAs. An additional parameter, the breeding ratio (Ψ), determines the proportion of the population which undergoes breeding (selection, crossover and mutation) in the current generation. Values for the breeding ratio parameter range from (0.0:1.0).

In each generation, after the fitness values of all the individuals in the same population are calculated, the bottom $(N \cdot \Psi)$, where N is the population size, are discarded and removed from the population. The remaining individual's velocity vectors are updated, acquiring new information from the population. The next generation is then created by updating the position vectors of these individuals to fill $(N \cdot (1 - \Psi))$ individuals in the next generation. The $(N \cdot \Psi)$ individuals needed to fill the population are selected from the individuals whose velocity is updated to undergo VPAC crossover and mutation and the process is repeated. For clarity, the flow of these operations is illustrated in Figure 1 where $k = (N \cdot (1 - \Psi))$.

The crossover operator used here was developed to utilize information available in the Breeding Swarm algorithm, but not available in the standard GA implementation. The new crossover operator, Velocity Propelled Averaged Crossover

Table 1: Table of parameters for GA, PSO and BS.

Parameter	GA	PSO	BS
Population size	50	50	50
Max iterations	2000	2000	2000
Initial Range	$[-0.5, 0.5]$	$[-0.5, 0.5]$	$[-0.5, 0.5]$
Velocity Range	$[-1.0, 1.0]$	N/A	$[-1.0, 1.0]$
Selection type	elitism & tournament	N/A	tournament
Tournament size	2	N/A	2
Crossover rate	0.6	N/A	N/A
Mutation Rate	0.1	N/A	0.1
Mutation Variance	$0.7 \rightarrow 0.4$	N/A	$1.0 \rightarrow 0.0$
Social	N/A	2.0	2.0
Inertia	N/A	$0.7 \rightarrow 0.4$	$0.7 \rightarrow 0.4$

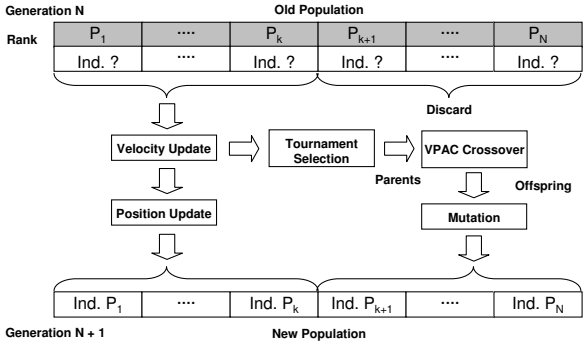


Figure 1: Flow of the Breeding Swarm Algorithm.

(VPAC), incorporates the PSO velocity vector. The goal is to create two new child particles whose position is between the parent's position, but accelerated away from the parent's current direction (negative velocity) in order to increase diversity in the population. Equations 3 show how the new child position vectors are calculated using VPAC. Towards the end of a typical PSO run, the population tends to be highly concentrated in a small portion of the search space, effectively reducing the search space. With the addition of the VPAC crossover operator, a portion of the population is always pushed away from the group, increasing the diversity of the population and the effective search space.

$$\begin{aligned} c_1(x_i) &= \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_1 p_1(v_i) \\ c_2(x_i) &= \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_2 p_2(v_i) \end{aligned} \quad (3)$$

Where, $c_1(x_i)$ and $c_1(x_i)$ are the positions of child 1 and 2 in dimension i, respectively. $p_1(x_i)$ and $p_2(x_i)$ are the positions of parents 1 and 2 in dimension i, respectively. $p_1(v_i)$ and $p_2(v_i)$ are the velocities of parents 1 and 2 in dimension i, respectively. φ is a uniform random variable in the range [0.0:1.0]. The child particles retain their parent's velocity vector, $c_1(\vec{v}) = p_1(\vec{v})$,$c_2(\vec{v}) = p_2(\vec{v})$.The previous best vector is set to the new position vector, restarting the child's memory, $c_1(\vec{p}) = p_1(\vec{x})$,$c_2(\vec{p}) = p_2(\vec{x})$.

The velocity and position update rules remain unchanged from the standard inertial implementation of the PSO. The social parameters are set to 2.0, inertia is linearly decreased from $0.7 \rightarrow 0.4$. A maximum velocity (V_{max})of ± 1 was allowed.

The breeding ratio was set to an arbitrary 0.5, with the expectation that the best results would be with an even mix of the GA and PSO. However, this need not be the case and other values for the breeding ratio may provide better results. All other parameters were kept consistent with the implementations of GA and PSO to remove any bias. Tournament selection, with a tournament size of 2, was used to select individuals as parents for crossover. The mutation operator used is Gaussian mutation, with mean 0.0 and variance reduced linearly each generation from 1.0 to 0.0. Each weight in the chromosome has probability of mutation 0.1. Parameters used are summarized in Table 1

4. RECURRENT NEURAL NETWORK

4.1 Network Architecture

The recurrent neural network model chosen for this research is a discrete-time, multi-layered and strongly connected recurrent neural network. At each time step, activation propagates through one layer of connections at a time until the output layer is reached and the output of the network as a whole is determined for that time step. The network is strongly connected in that every hidden node is connected to every other node (including itself) in the network, except input nodes, regardless of which layer the node is in. In this experiment the network contains a single input node and output node. A sample network with two hidden layers and two nodes per hidden layer is shown in Figure 2. The number of hidden layers and nodes per hidden layer is fixed for each experiment, but different configurations are tested in separate experiments.

Each node uses a symmetric sigmoidal activation function:.

$$f(x) = \frac{2}{1 + \exp(-\lambda x)} - 1 \quad (4)$$

Where λ is the slope of the symmetric sigmoidal function. For this experiment λ is set to 1.

At each time step the input node's activation level is set. Then for each hidden layer the activation level of each node in that layer is calculated. The activation levels for a layer are updated to the new values only after all of the activation

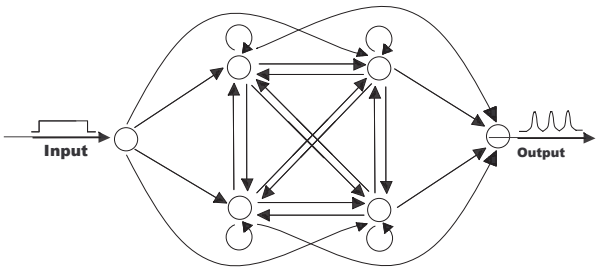


Figure 2: A sample strongly connected recurrent network with two hidden layers and two nodes per hidden layer.

Algorithm 1 Procedure for activating the recurrent neural network

```
Set all node values to zero.
for Each time step (1 to 100) do
    Set the input node value.
    for Each hidden layer do
        for Each hidden node in the layer do
            Calculate the activation level
        end for
        for Each hidden node in the layer do
            Update the activation level
        end for
    end for
    Calculate the output node's activation level
end for
Evaluate the training error
```

levels for that layer have been calculated. Then the process repeats for the next hidden layer. The algorithm for activating the recurrent neural network is given in Algorithm 1.

Effectively the activation levels of neurons in the same hidden layer are updated simultaneously and the activation levels of layers as a whole and updated sequentially.

4.2 Test Problem

A single biological neuron is capable of producing a wide variety of complex output patterns, for example pulsed outputs, on-responses and off-responses [3, 12, 18]. A recurrent neural network is needed in order to produce these complex output patterns from constant inputs.

The test problem chosen for this research is to evolve a multi-layer strongly connected recurrent neural network that has properties similar to those of biological neurons [19]. Our goal is to evolve a network that produces a simple pulsed output when a constant activation 'voltage' is applied. The test spans 100 time steps where the activation 'voltage' is 'turned on' at time step 10 and 'turned off' at time step 90 (i.e activation of the input node is set to 1 from time steps 10 to 90 and 0 at all other times). The output is a simple pulsed output with amplitude 1 and period 20, from time steps 10 to 90 (See figure 3).

There is no source of periodicity within the network, nodes, or input. Thus, the only way to produce the desired output pattern is to use the recurrent connections within the network. Further, the desired output period is fairly long (20

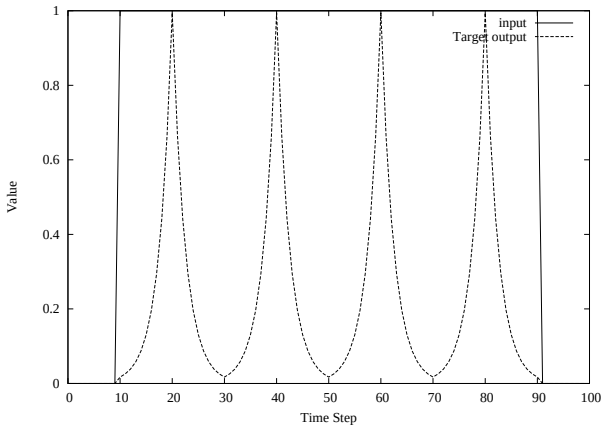


Figure 3: Expected target output signal for evolved network.

time steps). This means that typical deterministic training algorithms, such as those based on gradient descent, would find this problem almost impossible to solve. A gradient based algorithm would require unfolding the network at least 20 steps. Using a recurrent neural network reduces the size of the network to a more manageable size.

The error function used in these experiments is:.

$$E = \sum_{i=1}^{100} [(t_i - o_i)^2 + 2 \cdot (ts_i - os_i)^2] \tag{5}$$

where, t_i is the desired or *target* response value at time step i and o_i is the actual response of the network at time step i. The values ts_i and os_i represent the target and actual slope of the response at time step i. Where $ts_i = t_i - t_{i-1}$ and $os_i = o_i - o_{i-1}$. The slope component of the error function is meant to help steer the training towards periodic behavior.

4.3 Test Parameters

In order to verify that the results obtained using Breeding Swarm are acceptable, we compare the results to those obtained using both Genetic Algorithms and Particle Swarm Optimizers.

Of particular interest is how well the Breeding Swarm algorithm scales as the size of the neural network increases. We experiment with networks with a variety of hidden layers and nodes per hidden layer. In particular the number of hidden layers ranged from 1 to 9 and the number of nodes per hidden layer also ranged from 1 to 9 for a total of 80 different networks (excluding network size of 1 layer and 1 node per layer). For each network we ran 50 separate trials.

Each algorithm uses a vector of real values to represent the network weights. Each real value corresponds to a weight between one pair of nodes, or between a node and itself. The vector size varies depending on the size of the network. For a network with L hidden layers and N nodes per hidden layers there are a total of $(LN)^2 + 2LN$ weights, each of the LN hidden nodes is connected to every other hidden node and the input and output nodes are each connected to LN nodes. Thus, the vector consists of $(LN)^2 + 2LN$ real values.

Each algorithm run for 2000 generations on a population size of 50 particles/individuals. The initial weights were randomly chosen in the range [-0.5,0.5]. The velocity vector,

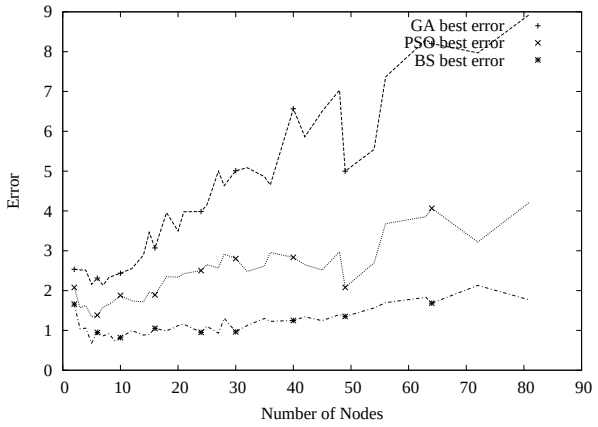


Figure 4: Average best error (averaged across all networks with the same number of nodes) values for each algorithm vs. total number of network nodes.

where applicable, was allowed to explore values in the range of [-1.0,1.0].

It is worth noting that no special care was taken in assigning the GA, PSO, or BS parameters and result shown are without optimizing the algorithms for this problem.

5. RESULTS

A t-test for significance was performed on each pair of algorithms ($\alpha = 0.05$). The PSO algorithm was found to perform significantly better than the GA algorithm in 73 of the 80 networks. The BS algorithm performed significantly better than the GA algorithm in 75 of 80 networks. The BS algorithm performed significantly better than the PSO algorithm in 47 of 80 networks. The PSO algorithm significantly outperformed the BS algorithm on 1 network (8 layers with 2 nodes per layer). The GA algorithm never outperformed the PSO or BS algorithm. In 79 of 80 network the BS algorithm was able to find a single better network than either the GA or PSO algorithms.

Figure 4 shows the average single best error of each algorithm by node count (the number of nodes in the network). The BS algorithm consistently performs much better than the other algorithms as the number of nodes increases. For Breeding Swarms the performance as the network size increased, remained almost constant, the error increased at a much slower rate than either GA or PSO.

Figure 5 shows the average mean error of each algorithm by node count. Again, the BS algorithm consistently outperforms the other algorithms as the number of nodes increases. In particular the BS algorithm mean error performed at an almost constant level, regardless of network size, up to a network size of 42 nodes where the mean began to increase. These results show that on average the BS algorithm will give better results than either GA or PSO.

Figure 6 shows the average single worst error of each algorithm by node count. The worst error found by the BS algorithm remained almost constant. In smaller networks the PSO algorithm's worst error performed better than either GA, or BS. In general the GA performed poorly on all network sizes.

Figures 7 through 9 show mean error values and best out-

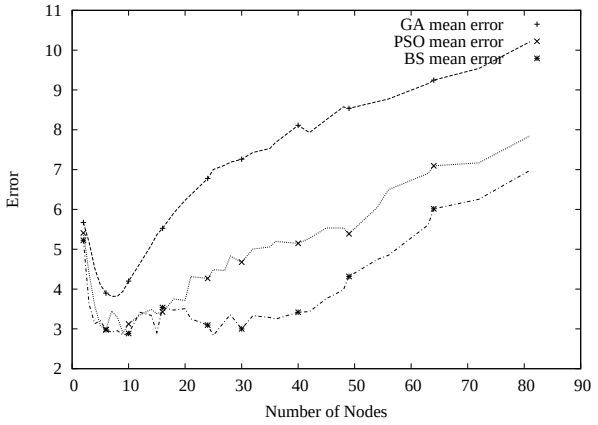


Figure 5: Average mean error (averaged across all networks with the same number of nodes) values for each algorithm vs. total number of network nodes.

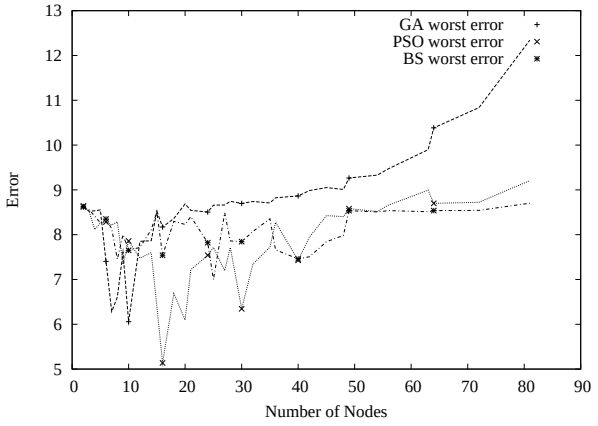


Figure 6: Average worst error (averaged across all networks with the same number of nodes) values for each algorithm vs. total number of network nodes.

puts for networks of sizes 5X1, 6X7 and 9X9 respectively. Figures 7(a), 8(a) and 9(a) show the average best error of each algorithm over time. Each graph shows that the BS algorithm behaves differently than both the GA and PSO. The GA and PSO algorithms behavior resembles that of an exponential, the fastest improvement occurs at the beginning of a run and levels off. Where in the BS the quickest improvement occurs towards the end of a run. This could be contributed to the VPAC crossover operator actively dispersing the population early in the run, maintaining diversity, and collapsing the population on a solution at the end of a run. During a typical BS run the velocity term steadily decreases. At the beginning of the run the velocity term is large and the population is pushed out, due to the VPAC operator, increasing diversity. Towards the end of a run the velocity term becomes small and the VPAC operator searches the mean of its parents, collapsing the population to a solution.

Figures 7(b), 8(b) and 9(b) show the best output obtained from the respective networks, it is clear that the BS algorithm finds a substantially better network than either the GA, or PSO algorithm. The Breeding Swarm algorithm was able to find solutions with higher peaks that more closely matched the desired output (smaller error). The GA algorithm was unable to find a network that had periodic behavior in the 9X9 network.

6. CONCLUSIONS

Training RNN to perform a certain task is a very difficult problem. Currently no ideal training algorithm exists. This research shows that the Breeding Swarms algorithm is a viable option when choosing an algorithm to train recurrent neural networks. The BS algorithm produced as good as, or better, results than the comparison algorithms in 79 of 80 test cases. As the network size increased the breeding swarm algorithm was able to scale better than the comparison algorithms, maintaining significantly better results for larger networks.

While the test problem chosen here is to train a recurrent neural network, the BS algorithm is a general population based algorithm that can be used to evolve solutions to a variety of problems. Future work includes analysis of the algorithm parameters and their effect on the behavior of the algorithm.

An additional advantage of the Breeding Swarm algorithm may be the ability to evolve network topologies and weights simultaneously, a trait not available in the standard PSO implementation. Through the crossover operator, a particle may be allowed to add new nodes or remove nodes, such as in GAs. Further research will look into this possibility.

7. ACKNOWLEDGMENTS

This work is supported by NSF EPSCoR EPS-0132626. The experiments were performed on a Beowulf cluster built with funds from NSF grant EPS-80935 and a generous hardware donation from Micron Technologies.

8. REFERENCES

[1] P. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto and et al., editors, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 601–610. Springer, 1998.

[2] P. Angeline, G. Saunders, and J. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, January 1994.

[3] J. Dowling. *Neurons and Networks: An Introduction to Neuroscience*. The Belknap Press of Harvard University Press, Cambridge, MA, 1992.

[4] R. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In e. a. V. William Porto, editor, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 611–616. Springer, 1998.

[5] M. El Choubassi, H. El Khoury, C. Jabra Alagha, J. Skaf, and M. Al-Alaoui. Arabic speech recognition using recurrent neural networks. In *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2003)*, Darmstadt, Germany, December 14–17 2003.

[6] L. Eshelman and J. Schaffer. Real-coded genetic algorithms and interval-schemata. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Mateo, CA, 1993.

[7] A. Esparcia-Alcazar and K. Sharman. Evolving recurrent neural network architectures by genetic programming. In J. R. Koza and et. al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 89–94, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[8] C. Giles, S. Lawrence, and A. Tsoi. Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, 44(1/2):161–183, July/August 2001.

[9] B. Hammer. *Learning with Recurrent Neural Networks*, volume 254. Springer, 2000.

[10] J. Holland. *Adaptation in Natural and Artificial Systems: 2nd ed.* University of Michigan Press, Ann Arbor, MI, 1992.

[11] B. Horne and C. Giles. An experimental comparison of recurrent neural networks. In G. Tesauro and et. al., editors, *In Advances in Neural Information Processing Systems*, volume 7, pages 697–704. The MIT Press, 1995.

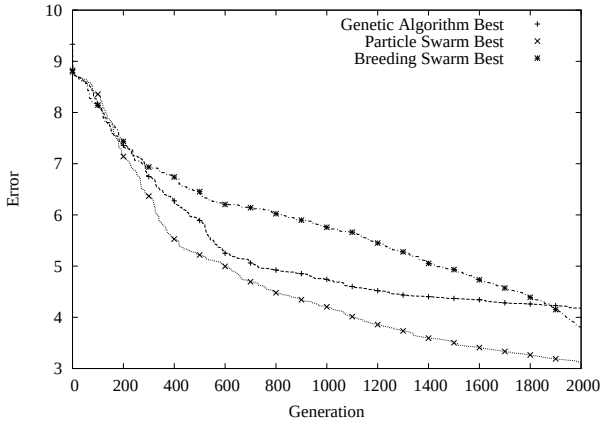
[12] E. Kandel and J. Schwartz. *Principles of Neuroscience, 2nd Edition*. Elsevier, New York, NY, 1985.

[13] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001.

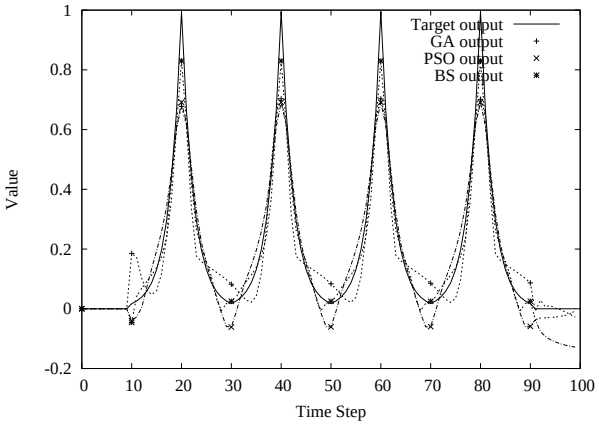
[14] M. Mandischer. Evolving recurrent neural networks with non-binary encoding". In *In Proc. Second IEEE Intl. Conf. Evelutionary Computation (ICEC '95)*, volume 2, pages 584–589, Perth, Australia, 1995. IEEE Press, Piscataway, NJ.

[15] K. Ohya. A sound synthesis by recurrent neural network. In E. Michie, editor, *Proceedings of the 1995 International Computer Music Conference*, pages 420–423, San Francisco: International Computer Music Association, 1995.

[16] G. Saunders, P. Angeline, and J. Pollack. Structural and behavioral evolution of recurrent networks. In

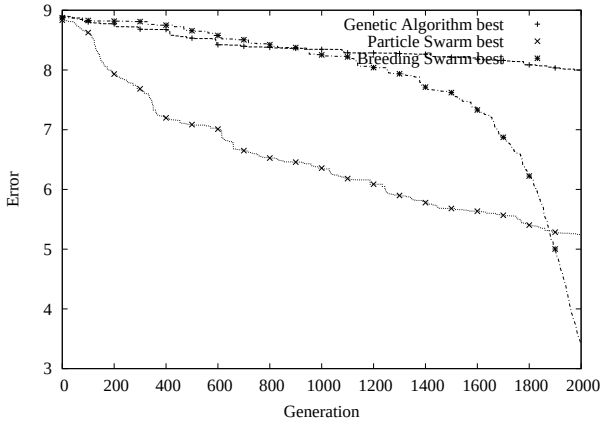


(a) Mean best error evaluation for each algorithm.

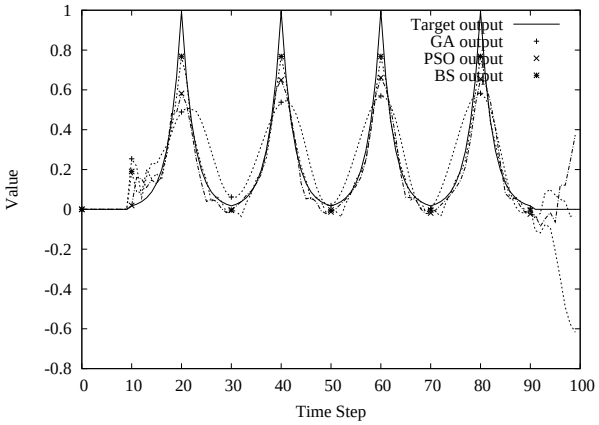


(b) Best output for each algorithm

Figure 7: Results for network with 5 layers and 1 node per layer (35 weights).

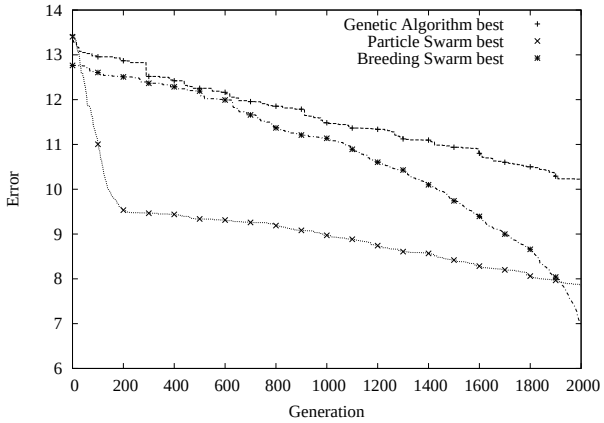


(a) Mean best error evaluation for each algorithm.

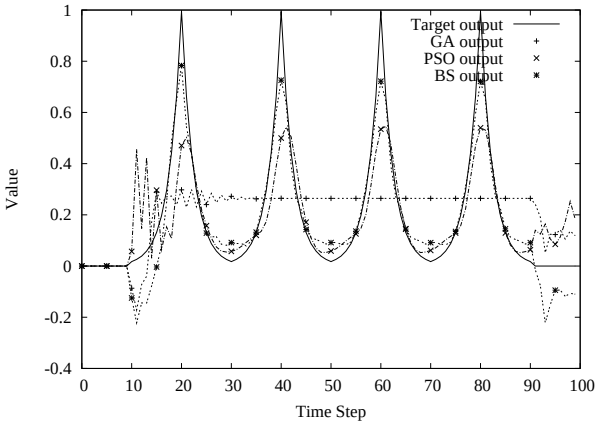


(b) Best output for each algorithm

Figure 8: Results for network with 6 layers and 7 nodes per layer (1848 weights).



(a) Mean best error evaluation for each algorithm.



(b) Best output for each algorithm

Figure 9: Results for network with 9 layers and 9 nodes per layer (6723 weights).

J. Cowan and et. al., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 88–95. Morgan Kaufmann Publishers, Inc., 1994.

[17] M. Settles, B. Rodebaugh, and T. Soule. Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In E. Cantú-Paz and et. al., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 148–149, Chicago, 12-16 July 2003. Springer-Verlag.

[18] G. Shepherd. *Neurobiology*. Oxford University Press, New York, NY, 1994.

[19] T. Soule, Y. Chen, and R. Wells. Evolving a strongly recurrent neural network to simulate biological neurons. In *In the proceedings of The 28th Annual Conference of the IEEE Industrial Electronics Society*, 2002.