Determining Neural Network Connectivity using Evolutionary Programming

John R. McDonnell and Don Waagen

NCCOSC, RDT&E Div.
San Diego, CA 92152-5000

ABSTRACT

This work investigates the application of evolutionary programming, a stochastic search technique, for determining connectivity in feedforward neural networks. The method is capable of simultaneously evolving both the connection scheme and the network weights. The number of synapses are incorporated into an objective function so that network parameter optimization is done with respect to a connectivity cost as well as mean pattern error. Experimental results are shown using feedforward networks for simple binary mapping problems.

INTRODUCTION

The neural network design process is largely based on heuristics. Previous experience (or the work of other researchers) often dictates an initial network configuration for the problem at hand. If the network can be trained to achieve the designer's goals, the design process is terminated. If success is not attained, a testing phase ensues and is largely trial and error. The result can often be a network with excess parameters and little regard for computational costs.

In this research, a connectivity cost associated with the neural network configuration is incorporated into the optimization procedure in an effort to reduce the number of synapses. An optimized architecture offers increased throughput for real-time signal processing applications as well as decreased memory requirements.

Simultaneously determining both network parameters and structure requires a search procedure which is amenable to combinatorial optimization. The more successful algorithms for these types of problems have generally been stochastic search techniques such as simulated annealing[1], genetic algorithms[2], and simulated evolution[3]. The simulated evolution, or evolutionary programming (EP), paradigm has been shown to have the desired attributes: combinatorial optimization capabilities[4], the ability to determine model structure[5], and the ability to train neural networks[6].

The premise of the current research is that near minimal size neural network architectures can be evolved under an objective function which incorporates both neural network connectivity and weight parameters. Further, the proposed approach takes advantage of computational resources during the design/training phase thereby removing the burden of evaluation by trial-and-error from the designer. For purposes of discussion, Fig. 1 illustrates the structure of a hypothetically evolved neural network where the connectivity between neurons is determined via a multi-agent stochastic search technique. Nodes which are not connected can be pruned. This work extends previous research in evolving neural network architectures (where both the number of neurons and connectivity are stochastically determined using EP[7]) by investigating an alternative strategy to evolving neural network connectivity.

Similar work has been undertaken by Bornholdt and Graudenz[8] using genetic algorithms to determine both network structure and parameters. Due to the generality of their implementation, recurrent networks can result requiring multiple sweeps to reach a stable state. The approach investigated in this work is limited to feedforward networks. The EP paradigm is outlined in the next section along with its application to training neural networks. This training method is then augmented so that the connectivity between layers can be randomly determined to yield a structure similar to that shown in Fig. 1. Finally, training results are given for simple binary mapping problems.

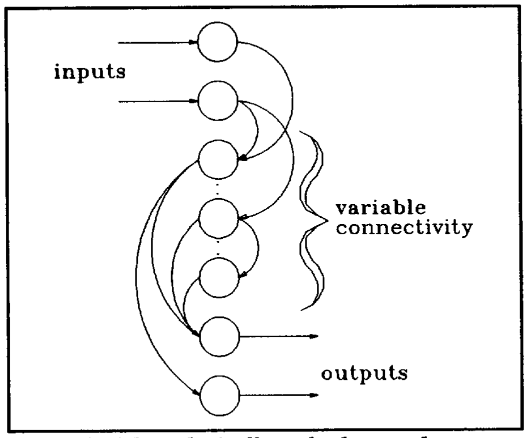

Figure 1. A hypothetically evolved network structure with variable connectivity.

APPLYING EP TO NEURAL NETS

Evolutionary Programming

Evolutionary programming is a neo-Darwinian search paradigm suggested by Fogel *et al.*[3] This stochastic search method is typically utilized as a global optimizer. EP has been successfully applied to a variety of optimization problems including the traveling salesman problem[4], parameter estimation and system identification[5], and neural net training[6].

The EP optimization algorithm can be described by the following steps:

*1. Form an initial population $P_{2N-1}(x)$ of size 2N. The parameters **x** associated with parent element P_i are randomly initialized from a user specified search domain.*
2. Assign a fitness score $S_i(x)$ to each element $P_i(x)$ in the population.
3. Reorder the population based on the number of wins generated from a stochastic competition process.
*4. Generate offspring (P_N .... P_{2N-1}) of the highest ranked N elements (P_0 .... P_{N-1}) in the population by perturbing **x**.*
5. Loop to step 2.

In addition to providing a systematic means of stochastic search, the generality of the EP optimization algorithm lends power to its implementation. The user is not bound to any particular coding structure nor mutation strategy. EP is used in this investigation since it is well suited for simultaneously evolving both model structure and parameters.

Determining Network Weights with EP

Evolutionary programming can be used for training neural networks. The selected objective function is the same as that used in backpropagation: minimize the sum-squared error function $E = ½\Sigma_p\Sigma_k(t_k - o_k)^2$ over all patterns p for k output neurons. The EP algorithm given in the previous section is applied to determining neural network weights and then results are shown for sample training runs using various scaling factors on the XOR mapping problem.

Initially, a population consisting of $2N$ feedforward networks is generated. Each network in the population is represented by a multidimensional weight array Φ_i with weights initially chosen from a $U[-0.5, 0.5]$ distribution. Next, a cost is assigned to each network in the population. This cost is typically the mean of the sum-squared pattern error E previously discussed. The "best" N members of the population generate offspring (perturbed weight sets) according to $W_o = W_p + \delta W_p$ where δW_p is $N(0, S_F \cdot E_p)$ with a scaling coefficient S_F and mean sum-squared pattern error E_p for each parent network. The scaling factor is a probabilistic analog to the stepsize used in gradient descent methods and may also be treated as a random variable within the EP search strategy[7]. The effect of the scaling factor is shown in Fig. 2 for the XOR mapping. The variance of the weight perturbations is bound by the total system error in this application. To emulate the probabilistic nature of survival, a pairwise competition is held where individual elements compete against randomly chosen members of the population. For example, if network Φ_j is randomly selected to compete against network Φ_i, a win is awarded to network Φ_i if $E_i < E_j$. The N networks with the most "wins" are kept and the process is repeated.

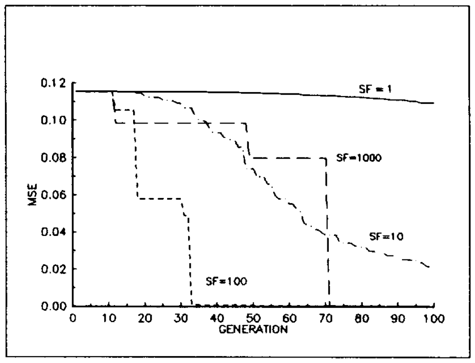

Figure 2. EP training of a 2-2-1 XOR mapping network for various scaling factors.

EVOLVING CONNECTIVITY

This section investigates structural level adaptation within the EP search. The objective function has been modified to be a linearly weighted combination of the number of connections N_c and the mean sum-squared pattern error

$$J = \alpha E + \beta N_c$$

A heuristic which might be employed would be to let $\beta \approx \alpha E_{crit}/N_{max}$ thereby incorporating the desired training error and the maximum possible number of connections N_{max} to reasonably weight the cost associated with the evolved number of connections.

Analogous to the weight array, a connectivity array has been specified where (one of its elements) $c = 1$ if a connection exists or $c = 0$ if no connection is present. A connectivity array that has all of its elements set to 1 yields a fully-connected feedforward network. The designer must specify the number of hidden neurons over which the search is conducted. This determines the maximum number of connections. In previous work[7], a synapse was randomly chosen from the range of possible connections and modified based on its current state. That is, disconnected synapses were connected and connected synapses were disconnected. The number of connections which may be affected at each mutation is arbitrarily set by the designer or may even be determined in a random fashion. The connectivity array is incorporated in the neuron output dot product term thereby nulling any signals between disconnected neurons. Weights are continually modified in the event that a neuron pair is reconnected.

In order to place an emphasis on signal propagation through the network, a strategy for connection and modification has been developed based on the activity levels of a neuron. This strategy assigns a probability of connection P_c to the connection between neuron j in layer l and neuron k in layer $l+1$, C_{ljk}, based upon the variance in neuron j 's output over all of the patterns n_p in the training set

$$P_c(C_{ljk}) = \frac{\sigma_{lj}^2}{\sum_{i \in l} \sigma_{li}^2}$$

where the variance for neuron j in layer l, σ_{lj}, is determined from the activation or output levels a_{lj} over the number of patterns n_p

$$\sigma_{lj}^2 = \frac{1}{(n_p - 1)} \sum_{n_p} (a_{lj} - \bar{a}_{lj})^2$$

Neurons which have high variance on their activation levels will tend to be connected to other neurons. This may also be viewed as promoting connections from neurons (which are essentially hyperplanes) that provide a measure of discrimination on the feature space. Neurons which have low variance on their activation levels correspond to hyperplanes which separate few data points, and thereby provide little information to the network.

Synapses are randomly chosen as candidates for modification. If a chosen synapse is not connected then it's probability of becoming connected is evaluated. Conversely, the probability of disconnection is calculated if the synapse is connected. If all of the synapses were candidates for mutation during each generation, this would be a self-fulfilling strategy where a single neuron would dominate. However, only a small number of randomly chosen synapses are evaluated for mutation when generating an offspring network. The probabilities of connection or disconnection are determined according to the neuron's class. The three classes are self-evident with the corresponding connection strategies as follows:

hidden unit neuron: The probability of connection is $P_c(C_{ljk})$ as calculated above, and the probability of disconnection is determined as $P_d(C_{ljk}) = (1 - P_c(C_{ljk}))$

input neuron: For the binary mappings used in these studies, the variance on the input units is constant and the strategy used for the input neurons reduces to a uniform probabilistic connection strategy previously used[7]. To promote coupling effects between neurons, the connection probabilities are multiplied yielding the probability of connection $P_c'(C_{ljk}) = P_c(C_{ljk}) * P_c(C_{(l+1)ki})$. The disconnection probability is determined in a similar fashion according to $P_d'(C_{ljk}) = P_d(C_{ljk}) * P_d(C_{(l+1)ki})$.

bias neuron: Since the bias neurons are invariant, the probability of connecting to any neuron is contingent upon the variance in the activation levels of that neuron. As a result, the probability of connecting to a given neuron is simply the probability of that neuron connecting to the next layer. Thus the connection probability of a bias neuron can be given by $P_c(C_{ljk}) = P_c(C_{(l+1)ki}))$. Likewise, its disconnection probability is given by $P_d(C_{ljk}) = P_d(C_{(l+1)ki})$.

RESULTS

Experiments were conducted with $N=10$ parent networks, $\alpha=1, \beta=0.001$, and $S_F=100$ for the XOR and 3-bit parity mappings using the probabilistic connection criteria discussed above. The networks were initialized with a random connection strategy as opposed to initially being fully connected. Figures 3-6 show the results of using this criteria for the XOR mapping with 8 hidden units. Figures 7 and 8 show an example of an evolved network for the 3 bit parity problem with 16 hidden units.

As a measure of the sparseness of a network's connectivity, a dilution ratio has been defined[8] as $D=S/N^2$ where S is the total number of synapses and N is the number of neurons. The dilution ratio for the evolved networks is given in Figures 4,6, & 8, respectively. The stochastic nature of the search process makes the advent of two exact evolved configurations highly unlikely. This is not to say that, once the unused neurons are discarded, the networks will always be dissimilar.

When compared to the purely random selection process previously implemented[7], the probabilistic approach developed in this investigation tends to yield networks with fewer connections at the expense of requiring knowledge of the network structure (the activation-level based connection scheme must know what class of neuron is being connected). The cost function can be modified[7] to incorporate the additional cost associated with the number of neurons.

CONCLUSIONS

Evolutionary programming can be used to simultaneously determine both network architecture and parameters. The results indicate that EP has high potential for automating the design of neural networks.

This method for reducing the number of redundant connections based on the variance of a neuron's activation level also appears promising. Some applications may be hindered by the excess memory required by multi-agent searches or the training time necessary to obtain adequate convergence. If the design process for a specific problem cannot be fully automated, this technique may provide a designer with insight as well as a reasonable starting point for further investigations.

Since only binary mapping problems were investigated, it is not clear how the approach given in this study will work on classification or continuous mapping problems. Nevertheless, stochastic training techniques are becoming prevalent in neurocomputing (especially in hardware implementations[9]). During these investigations, issues in orthogonal learning (search) and population dynamics became prevalent. These topics are being addressed in future work[10].

REFERENCES

1. E. Aarts and J. Korst, *Simulated Annealing and Boltzman machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, 1989.

2. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

3. L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley and Sons, 1966.

4. D. B. Fogel, "An evolutionary approach to the traveling salesman problem", *Biological Cybernetics*, Vol. 60, No. 2, 1988.

5. D. B. Fogel, *System Identification through Simulated Evolution: a Machine Learning Approach to Modeling*, Ginn Press, Needham, MA., 1991.

6. D.B. Fogel, L.J. Fogel and V.W. Porto, "Evolving neural networks", *Biological Cybernetics*, Vol. 63, 1990.

7. J. R. McDonnell and D. Waagen, "Evolving neural network architecture", SPIE Proc., Vol. 1766, Neural and Stochastic Methods in Image and Signal Processing, San Diego, 1992.

8. S. Bornholdt and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms", *Neural Networks*, Vol. 5, 1992.

9. B. W. Lee and B. J. Shen, "Analysis and design of analog VLSI neural networks", in *Neural Networks for Signal Processing*, B. Kosko (Ed.), Prentice-Hall, 1992.

10. J. R. McDonnell and D. Waagen, "Issues in evolving neural networks: orthogonal learning and population dynamics", submitted to SPIE Conf. on Science of Artificial Neural Networks II, 1993.

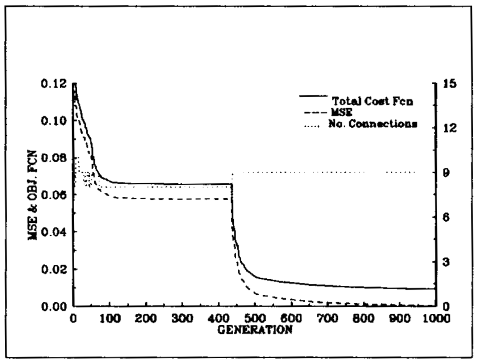

Figure 3. Evolving connectivity for the XOR mapping. See final architecture at right.

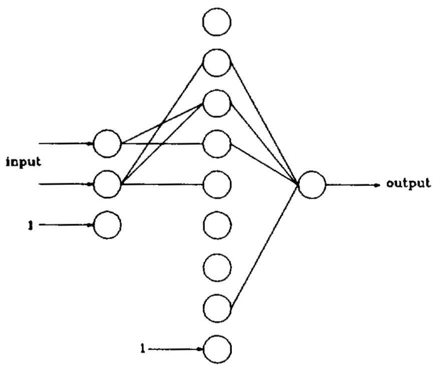

Figure 4. The final evolved connectivity for the XOR mapping network, D = 0.111.

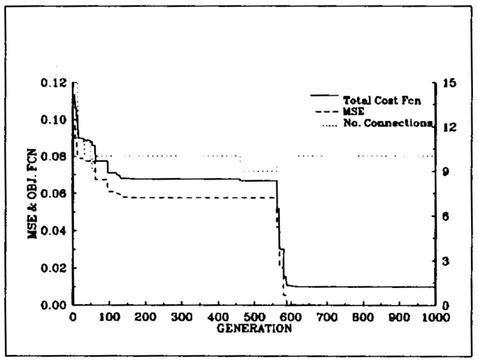

Figure 5. Evolving the connectivity for the XOR mapping. See final architecture at right.

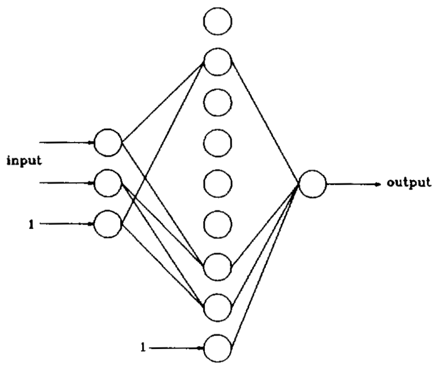

Figure 6. The final evolved connectivity for the XOR mapping network, D = 0.129.

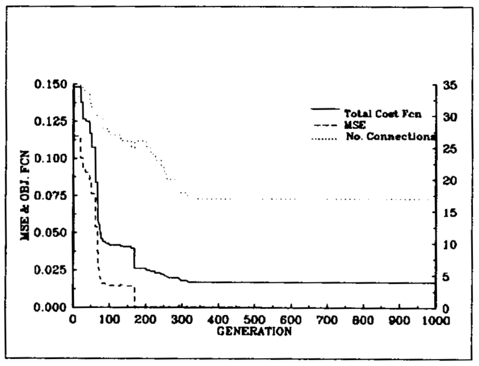

Figure 7. Evolving connectivity for the 3 bit parity mapping. See final architecture at right.

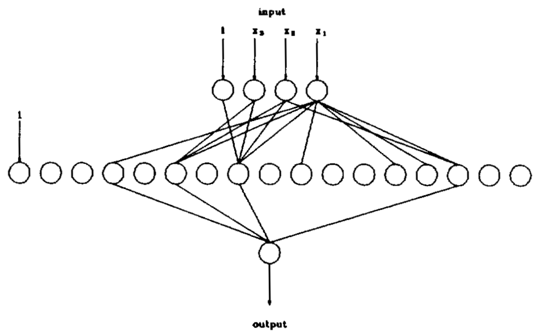

Figure 8. The final evolved connectivity for the 3 bit parity mapping network, D = 0.066.