Simultaneous Evolution of Architectures and Connection Weights in ANNs

Hussein A. Abbass [1] and Ruhul A. Sarker [2]

School of Computer Science,
University of New South Wales,
Australian Defence Force Academy Campus
Northcott Drive, Canberra 2600, Australia,

(E-mail : [1]h.abbass@adfa.edu.au, [2]r.sarker}@adfa.edu.au)

Abstract

Feed-forward Artificial Neural Networks (ANNs) have become popular among researchers and practitioners for modelling complex real–world problems. One of the latest research areas in this field is evolving ANNs. In this paper, we investigate the simultaneous evolution of architectures and connection weights in ANNs. In simultaneous evolution, we use the concept of multiobjective optimization and subsequently evolutionary multiobjective algorithms to evolve ANNs. The results are promising when compared with the traditional Backpropagation algorithm.

Keywords: Evolutionary Artificial Neural Networks, Differential Evolution

1.Introduction

Feed-forward Artificial Neural Networks (ANNs) have found extensive acceptance in many disciplines for modelling complex real–world problems. An ANN is formed from a group of units, called neurons or processing elements, connected with arcs, called synapses or links, where each arc is associated with a weight representing the strength of the connection, and usually the nodes are organized in layers. Each neuron has an input equals to the weighted some of the outputs of those neurons connected to it. The weighted sum of the inputs represents the activation of the neuron. The activation signal is passed through a transfer function to produce a single neuron's output. The transfer function introduces non–linearity to the network. The behavior of a neural network is determined by the transfer functions, the learning rule by which arcs update their weights, and the architecture itself in terms of the number of connections and layers. Training is the process of adjusting the networks' weights to minimize the difference between the network output and the desired output on a suitable metric space. Once the network is trained, it can be tested by a new dataset.

As previously mentioned, the performance of a neural network for a given problem, depends on the transfer function, network architecture, connection weights, inputs, and learning rule. The architecture of an ANN includes its topological structure, i.e., connectivity and number of nodes in the network. The architecture design is crucial for successful application of ANNs because the architecture has a significant impact on the overall processing capabilities of the network. In most function approximation problems, one hidden layer is sufficient to approximate continuous functions (Basheer 2000; Hecht-Nielsen 1990). Generally, two hidden layers may be necessary for learning functions with discontinuities (Hecht-Nielsen 1990). The determination of the appropriate number of hidden layers and number of hidden nodes in each layer is one of the important tasks in ANN design. A network with too few hidden nodes would be incapable of differentiating between complex patterns leading to only a lower estimate of the actual trend. In contrast, if the network has too many hidden nodes it will follow the noise in the data due to over-parameterization leading to poor generalization for test data (Basheer and Hajmeer 2000). With increasing number of hidden nodes, training becomes excessively time-consuming.

The most popular approach to finding the optimal number of hidden nodes is by trial and error. Methods for network growing (such as cascade correlation (Fahlman and Lebiere 1990)) and for network pruning (such as Optimal brain damage (LeCun, Denker, and Solla 1990)) have been used to overcome the unstructured and somehow unmethodical process for determining a good network architecture. However, all these methods still suffer from their slow convergence and long training time. Nowadays, many researchers use evolutionary algorithms to find the appropriate network architecture by minimizing the output error (Kim and Han 2000; Yao and Liu 1998).

Weight training in ANNs is usually formulated as minimization of an error function, such as the mean square error between target and actual outputs averaged over all examples (training data), by iteratively adjusting connection weights. Most training algorithms, such as *Backpropagation* (BP) and conjugate gradient are based on gradient descent (Basheer and Hajmeer 2000; Hertz, Krogh, and Palmer 1991). Although BP has some successful applications, the algorithm often gets trapped in a local minimum of the error function and is incapable of finding a global minimum if the error function is multimodal and/or non–differentiable (Yao 1999). To overcome this problem, one can use evolutionary algorithms for weight training. The application of evolutionary algorithms for weight training can be found in (Kim and Han 2000; Yao and Liu 1998).

As discussed above, it is necessary to determine the appropriate architecture of the network and appropriate connection weights to ensure the best performance out of ANNs. Most researchers treat them as two independent optimization problems. As Yao (1999) indicated, connection weights have to be learned after a near-optimal architecture is found. This is especially true if one uses the indirect encoding scheme, such as the developmental rule method. One major problem with the determination of architectures is noisy fitness evaluation (Yao and Liu 1997). In order to reduce such noise, an architecture usually has to be trained many times from different random initial weights. This method increases the computational time for fitness evaluation dramatically. If we look at the theoretical side of such optimization problems, this sequential optimization procedure (first architecture optimization followed by weight optimization) will usually provide a suboptimal solution for the overall problem.

To overcome this problem, the natural choice is to determine the architecture and connection weights simultaneously by solving a single optimization problem with two objectives. Many researchers attempted to ignore the architecture and minimize only the mean sum square error function (Kim and Han 2000; Yao and Liu 1998). A comprehensive list of papers on this topic can be found in (Yao 1999). However, if the contribution – to the objective function – of a sub-problem is very low compared to the other, the effect of the first sub-problem will not be reflected properly in the overall optimal solution. In such situations, simultaneous multiobjective optimization would be a better choice.

The purpose of this research is to determine the architecture and connection weights of ANNs simultaneously by treating the problem as a multiobjective optimization problem. We believe the simultaneous Evolution of Architectures and Connection Weights in ANNs using the concept of multiobjective optimization would add a new direction of research in ANNs. In addition, we will show experimentally that this approach performs better than BP with much lower computational cost (usually proportion to the maximum number of hidden units needed to be tested.

The paper is organized as follows. After introducing the research context, multiobjective optimization and the proposed algorithm are introduced in Section 2 followed by a case study on simultaneous evolution in Section 3. Experiments are then presented in Section 4 and conclusions are drawn in Section 5.

2.Multiobjective Optimization

In multiobjective optimization, all the objectives must be optimized concurrently to get the real trade-off for decision making. In this case, there is no single optimal solution, but rather a set of alternative solutions. These solutions are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered. They are known as pareto-optimal solutions.

There are several conventional optimization based algorithms for solving multiobjective optimization problems (Coello, 1999). These methods are not discussed here since none of them perform simultaneous optimization. Evolutionary algorithms (EAs) seem to be particularly suited for multiobjective optimization problems because they process a set of solutions in parallel, possibly exploiting similarities of solutions by recombination. Some researchers suggest that multiobjective search and optimization might be a problem area where EAs do better than other blind search strategies (Fonseca and Fleming, 1995 and Valenzuela-Rendon and Uresti-Charre, 1997). There are several EAs available in the literature that are capable of searching for multiple pareto-optimal solutions concurrently in a single run. Some of the popular algorithms are: the Vector Evaluated Genetic Algorithm (VEGA) (Schaffer 1985), Hajela's and Lin's genetic algorithm (HLGA) (Hajela and Lin 1992), Nondominated Sorting Genetic Algorithms (NSGA) (Srinivas and Dev 1994), Fonseca and Fleming's evolutionary algorithm (FFES) (Fonseca and Fleming 1993), Niched Pareto Genetic Algorithm (NPGA) (Horn, Nafpliotis, and Goldberg 1994), the Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler and Thiele 1999), and the Pareto Archived Evolution Strategy (PAES) (Knowles and Corne 1999; Knowles and Corne 2000). However, none of these algorithms performs consistently for all types of problems. Recently, we developed the *Pareto-based Differential Evolution* (PDE) approach, which outperforms most existing evolutionary multi-objective algorithms over continuous domains (Abbass, Sarker, and Newton 2001).

2.1.Differential Evolution

Evolutionary algorithms (Fogel 1995) is a kind of global optimization techniques that use selection and recombination as their primary operators to tackle optimization problems. DE is a branch of evolutionary algorithms developed by Rainer Storn and Kenneth Price (Storn and Price 1995) for optimization problems over continuous domains. In DE, each variable is represented in the chromosome by a real number. The approach works as follows:-

1. Create an initial population of potential solutions at random, where it is guaranteed, by some repair rules, that variables' values are within their bound-

aries.

2. Until termination conditions are satisfied
   (a) Select at random a trail individual for replacement, an individual as the main parent, and two individuals as supporting parents.
   (b) With some probability, each variable in the main parent is perturbed by adding to it a ratio, F, of the difference between the two values of this variable in the other two supporting parents. At least one variable must be changed. This process represents the crossover operator in DE.
   (c) If the resultant vector is better than the trial solution, it replaces it; otherwise the trial solution is retained in the population.
   (d) go to 2 above.

From the previous discussion, DE differs from *genetic algorithms* (GA) in a number of points:

1. DE uses real number representation while conventional GA uses binary, although GA sometimes uses integer or real number representation as well.
2. In GA, two parents are selected for crossover and the child is a recombination of the parents. In DE, three parents are selected for crossover and the child is a perturbation of one of them.
3. The new child in DE replaces a randomly selected vector from the population only if it is better than it. In conventional GA, children replace the parents with some probability regardless of their fitness.

2.2.A Differential Evolution algorithm for MOPs

A generic version of the adopted algorithm can be found in (Abbass, Sarker, and Newton 2001). The PDE algorithm is similar to the DE algorithm with the following modifications:-

1. The initial population is initialized according to a Gaussian distribution $N(0.5, 0.15)$.
2. The step-length parameter is generated from a Gaussian distribution $N(0, 1)$.
3. Reproduction is undertaken only among non-dominated solutions in each generation.
4. The boundary constraints are preserved either by reversing the sign if the variable is less than 0 or keeping subtracting 1 if it is greater than 1 until the variable is within its boundaries.
5. Offspring are placed into the population if they dominate the main parent.

The algorithm works as follows. An initial population is generated at random from a Gaussian distribution with mean 0.5 and standard deviation 0.15. All dominated solutions are removed from the population. The remaining non-dominated solutions are retained for reproduction. A child is generated from a selected three parents and is placed into the population if it dominates the first selected parent; otherwise a new selection process takes place. This process continues until the population is completed.

3.Proposed Algorithm

3.1.Nomenclatures

From herein, the following notations will be used for a single hidden layer MLP:

- I and H are the number of input and hidden units respectively.
- $\mathbf{X}^p \in \mathbf{X} = (x_1^p, x_2^p, \ldots, x_I^p), p = 1, \ldots P$, is the pth pattern in the input feature space $\mathbf{X}$ of dimension I, and P is the total number of patterns.
- Without any loss of generality, $\mathbf{Y}_o^p \in \mathbf{Y_o}$ is the corresponding scalar of pattern $\mathbf{X}^p$ in the hypothesis space $\mathbf{Y_o}$.
- w_{ih} and w_{ho}, are the weights connecting input unit $i,\ i = 1 \ldots I$, to hidden unit $h,\ h = 1 \ldots H$, and hidden unit h to the output unit o (where o is assumed to be 1 in this paper) respectively.
- $\Theta_h(\mathbf{X}^p) = \sigma(a_h); a_h = \sum_{i=0}^{I} w_{ih} x_i^p,\ \ h = 1 \ldots H$, is the h^{th} hidden unit's output corresponding to the input pattern $\mathbf{X}^p$, where a_h is the activation of hidden unit h, and $\sigma(.)$ is the activation function that is taken in this paper to be the logistic function $\sigma(z) = \frac{1}{1+e^{-Dz}}$, with D the function's sharpness or steepness and is taken to be 1 unless it is mentioned otherwise.
- $\hat{Y}_o^p = \sigma(a_o); a_o = \sum_{h=0}^{H} w_{ho} \Theta_h(\mathbf{X}^p)$ is the network output and a_o is the activation of output unit o corresponding to the input pattern $\mathbf{X}^p$.

3.2.Representation

In deciding on an appropriate representation, we tried to choose a representation that can be used for other architectures without further modifications. Our chromosome is a class that contains one matrix Ω and one vector ρ. The matrix Ω is of dimension $(I + O) \times (H + O)$. Each element $\omega_{ij} \in \Omega$, is the weight connecting unit i with unit j, where $i = 0, \ldots, (I - 1)$ is the input unit i, $i = I, \ldots, (I + O - 1)$ is the output unit

$i - I$, $j = 0, \ldots, (H-1)$ is the hidden unit j, and $j = H, \ldots, (H+O-1)$ is the output unit $j - H$. This representation has the following two characteristics that we are not using in the current version but can easily be incorporated in the algorithm for future work:-

1. It allows direct connection from each input to each output units (we allow more than a single output unit in our representation).
2. It allows recurrent connections between the output units and themselves.

The vector ρ is of dimension H, where $\rho_h \in \rho$ is a binary value used to indicate if hidden unit h exists in the network or not; that is, it works as a switch to turn a hidden unit on or off. The sum, $\sum_{h=0}^{H} \rho_h$, represents the actual number of hidden units in a network, where H is the maximum number of hidden units. This representation allows both training the weights in the network as well as selecting a subset of hidden units.

3.3.Methods

We have a function approximation problem which may arise in many situations including data mining, forecasting and estimation. We have no prior knowledge about the nature of the function. Based on the discussions in the first section, we have decided to use one input layer, one hidden layer and one output layer in the network. Our main objective is to estimate the connection weights by minimizing the total error, and select the appropriate number of hidden nodes. In this paper, we need to determine the connection weights which are real variables and select the hidden nodes in the network which are associated each with a binary variable (1 if the hidden unit exists and 0 for not). We set two objectives in this problem as follows:

1. Minimization of error
2. Minimization of number of hidden units in ANN

The problem can be handled as a multiobjective Optimization Problem. The steps to solve this problem are given below.

1. Create a random initial population of potential solutions.
2. Until termination conditions are satisfied, repeat
    (a) Evaluate the individuals in the population and label those who are non-dominated.
    (b) Delete all dominated solutions from the population.
    (c) Repeat
        i. Select at random an individual as the main parent and two individuals as supporting parents.
        ii. With some probability $Uniform(0,1)$, crossover the parents to generate a child where each weight in the main parent is perturbed by adding to it a ratio, $F \in Gaussian(0,1)$, of the difference between the two values of this variable in the two supporting parents. At least one variable must be changed.
        iii. If the child dominates the main parent, place it into the population.
    (d) Until the maximum population size is reached

4.Experiments

4.1.Experimental Setup

To test the performance of our proposed method, we experimented with a known polynomial function in two variables and of the third degree with noise. Noise was added to each input with a probability 0.2 from a Gaussian distribution $N(0, 0.2)$. The function took the form

$$x_1^3 + x_2^3$$

We generated 2000 instances as a training set and another 2000 as a test set. Both training and test sets were generated independently. Variables were generated from a uniform distribution between 0 and 1. After computing the output of the function, noise were added to the inputs only.

For the evolutionary approach, an initial population size of 50 was used and the maximum number of objective evaluations was set to 20,000. The number of inputs and the maximum number of hidden nodes were chosen as 2 and 10 respectively. The PDE algorithm was run with five different crossover rates (0, 0.05, 0.1, 0.5 and 0.9) and five different mutation rates (0, 0.05, 0.1, 0.5 and 0.9). For each combination of crossover and mutation rates, results were collected and analyzed over 10 runs with different seed initializations. The initial population is initialized according to a Gaussian distribution $N(0,1)$. The step-length parameter F is generated for each variable from a Gaussian distribution $N(0,1)$. The algorithm is written in standard C^{++} and ran on a Sun Sparc 4.

The Backpropagation algorithm was tested for ten different architectures created varying the hidden nodes from 1 to 10. For each architecture, the Backpropagation algorithm was run 10 times with ten different random seeds. The same ten seeds were used in all BP runs as well as the evolutionary approach.

4.2.Experimental Results and Discussions

For each architecture, the results from each of the 10 runs of the Backpropagation algorithm were recorded and analyzed. The performance of the evolutionary approach is measured by the average performance of the pareto set, which is selected on the training set, on the test set. The average error rate from these 100 runs (10 architectures, each with 10 runs) is found to be 0.095 with a range of architecture-wise average 0.094 to 0.096. The average error rates with different crossover and mutation rates for the PDE approach are plotted in Figures 1 and 2. Each average error rate is a mean of ten runs for a given crossover and mutation. In the x-axis of both figures, the numbers 1 to 5 represent the crossover or mutation rates from 0.0 to 0.9 as discussed earlier.

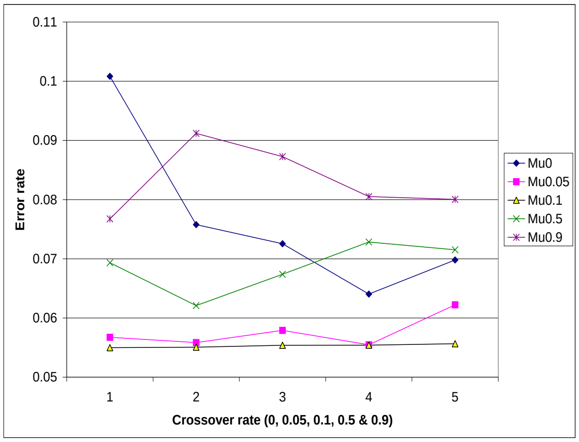

Figure 1: Error rate vs. crossover rate

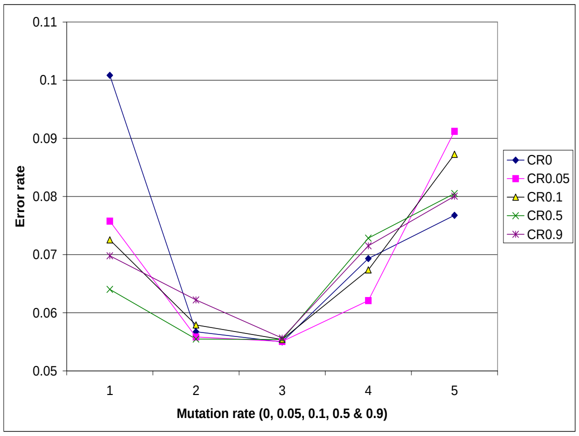

Figure 2: Error rate vs. mutation rate

As we see in Figure 1, the error rate is minimum when the mutation rate is 0.1. At this mutation rate, the error rate varies within a narrow zone of 0.055 to 0.056. As shown in Figure 2, for all crossover rates, the error rate versus mutation rate follows a convex-like curve. Here the error rate decreases up to the minimum, where mutation rate is 0.10, and then increases. This nice pattern helps in choosing the optimum crossover and mutation rates. In our case, a mutation rate of 0.10 and a crossover rate of 0.05 achieved the best performance with an Error rate of 0.055. With this best crossover and mutation rates, the best error rate in a single run is 0.054 with three hidden nodes (a solution from the Pareto front).

Taking the average figures in both Backpropagation and PDE approaches, we can see that the PDE approach reduces the error rate from 0.095 to 0.055 which is around 42 percent improvement. This emphasizes the advantages of the evolutionary approach in terms of accuracy and speed.

We need to emphasize here that the evolutionary approach performed in the same number of epochs better than what 10 different BP runs did. To explain this further, we needed to find the best number of hidden units on our test task. To do this, we trained 10 different neural networks with number of hidden units ranging from 1 to 10. In the evolutionary approach, however, we sat the maximum number of hidden units and the evolutionary approach determined the appropriate number without the need of experimenting with 10 different networks. In addition, the crossover is much faster than BP, adding more advantages to the evolutionary approach.

5.Conclusions and Future Research

In this research, we investigated the simultaneous evolution of architectures and connection weights in ANNs. In so doing, we proposed the concept of multiobjective optimization to determine the best architecture and appropriate connection weights concurrently. The multiobjective optimization problem was then solved using the Pareto Differential Evolution algorithm. The results on a test problem was significantly better when compared with Backpropagation. Although it shows a very promising performance, in our future work, we will need to experiment with more problems to generalize our findings.

6.Acknowledgment

This work is supported with ADFA Special Research Grants TERM6 2001 DOD02 ZCOM Z2844.

References

Abbass, H., R. Sarker, and C. Newton (2001). A pareto differential evolution approach to vector optimisation problems. *Congress on Evolutionary Computation 2*, 971–978.

Basheer, I. (2000). Selection of methodology for modeling hystersis behaviour of soils using neural networks. *J. Comput.-Aided Civil Infrastruct. Eng. 5*, 445–463.

Basheer, I. and M. Hajmeer (2000). Artificial neural networks: Fundamentals, computing, design, and applications. *Journal of Microbiological Methods 43*, 3–31.

Fahlman, S. and C. Lebiere (1990). The cascade correlation learning architecture. Technical Report CMU-CW-90-100, Canegie Mellon University, Pittsburgh, PA.

Fogel, D. (1995). *Evolutionary Computation: towards a new philosophy of machine intelligence*. IEEE Press, New York, NY.

Fonseca, C. and P. Fleming (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California*, 416–423.

Hajela, P. and C. Lin (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization 4*, 99–107.

Hecht-Nielsen, R. (1990). Neurocomputing. *Addison-Wesley, Readings, MA, USA*.

Hertz, J., A. Krogh, and R. Palmer (1991). Introduction to the theory of neural computation. *Reading, MA: Addison-Wesley*.

Horn, J., N. Nafpliotis, and D. Goldberg (1994). A niched pareto genetic algorithm for multiobjective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation 1*, 82–87.

Kim, K. and I. Han (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications 19*, 125–132.

Knowles, J. and D. Corne (1999). The pareto archived evolution strategy: a new baseline algorithm for multiobjective optimization. *In 1999 Congress on Evolutionary Computation, Washington D.C., IEEE Service Centre*, 98–105.

Knowles, J. and D. Corne (2000). Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation 8*(2), 149–172.

LeCun, Y., J. Denker, and S. Solla (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems*. Morgan Kaufmann.

Schaffer, J. (1985). Multiple objective optimization with vector evaluated genetic algorithms. *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, 93–100.

Srinivas, N. and K. Dev (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation 2*(3), 221–248.

Storn, R. and K. Price (1995). Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE 87*, 1423–1447.

Yao, X. and Y. Liu (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Trans. on Neural Networks 8*, 694–713.

Yao, X. and Y. Liu (1998). Making use of population information in evolutionary artificial neural networks. *IEEE Trans. on Systems, Man and Cybernetics 28*, 417–425.

Zitzler, E. and L. Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*(4), 257–271.