International Journal of Electronics and Communication Engineering.
ISSN 0974-2166 Volume 4, Number 5 (2011), pp. 517-525
© International Research Publication House
http://www.irphouse.com

Evolutionary Algorithm for Connection Weights in Artificial Neural Networks

[1]G.V.R. Sagar and [2]Dr. S. Venkata Chalam

[1]Assoc. Professor,
G.P.R. Engg. College (Autonomous), Kurnool, A.P., India
E-mail: nusagar@gmail.com
[2]Principal, A C E Engineering College, Ghatkesar, R.R. District, AP, India
E-mail: sv_chalam2003@yahoo.com

Abstract

A neural network may be considered as an adaptive system that progressively self-organizes in order to approximate the solution, making the problem solver free from the need to accurately and unambiguously specify the steps towards the solution. Moreover, Evolutionary Artificial Neural Networks (EANNs) have the ability to progressively improve their performance on a given task by executing learning. An evolutionary computation gives adaptability for connection weights using feed forward architecture. In this paper, the use of evolutionary computation for feed-forward neural network learning is discussed. To check the validation of proposed method, XOR benchmark problem has been used. The accuracy of the proposed model is more satisfactory as compared to gradient method.

Keywords: Evolutionary algorithm, Gradien decent, Back-Propagation, Mean square error.

Introduction

Architectures: An ANN consists of a set of processing elements, also known as neurons or nodes, which are interconnected.

It can be described as a directed graph in which each node performs a transfer function of the form

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij} x_j - \Theta_I \right) \tag{1}$$

where y_i is the output of the node i, x_j is the th input to the node, and is the connection weight between nodes i and j, Θ_i. is the threshold (or bias) of the node.

Learning in ANN's: Learning is achieved by adjusting the connection weights in ANN's iteratively so that trained (or learned) ANN's can perform certain tasks. Learning in ANN's can roughly be divided into supervised, unsupervised, and reinforcement learning. Supervised learning is based on direct comparison between the actual output of an ANN and the desired correct output, also known as the target output. It is often formulated as the minimization of an error function such as the total mean square error between the actual output and the desired output summed over all available data.

Evolutionary Artificial Neural Network

Evolutionary artificial neural networks (EANN's) refer to a special class of artificial neural networks (ANN's) in which evolution is another fundamental form of adaptation in addition to learning [2] – [5]. Evolutionary algorithms (EA's) are used to perform various tasks, such as connection weight training, architecture design, learning rule adaptation, input feature selection, connection weight initialization, rule extraction from ANN's, etc. One distinct feature of EANN's is their adaptability to a dynamic environment. The two forms of adaptation, i.e., evolution and learning in EANN's, make their adaptation to a dynamic environment much more effective and efficient.

Evolution has been introduced into ANN's at roughly three different levels: connection weights, architectures, and learning rules. The evolution of connection weights introduces an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigm where gradient-based training algorithms often experience great difficulties. The evolution of architectures enables ANN's to adapt their topologies to different tasks without human intervention and thus provides an approach to automatic ANN design as both ANN connection weights and structures can be evolved.

Feed forward ANN Architecture

ANN's can be divided into feed-forward and recurrent classes according to their connectivity. An ANN is feed-forward if there exists a method which numbers all the nodes in the network such that there is no connection from a node with a large number to a node with a smaller number. The feed-forward neural networks allow only for one directional signal flow. Furthermore, most of feed-forward neural networks are organized in layers. An example of the three layer feed-forward neural network is shown in fig.2.0

This network consists of input nodes, two hidden layers and an output layer. Typical activation functions are shown in fig.2.1. These continuous activation functions allow for the gradient based training of multilayer networks. A single neuron can divide only linearly separated patterns. In order to select just one region in n-dimensional input space more than n+1 neurons should be used. If more input clusters should be selected than the number of neurons in the input(hidden) layer should be properly multiplied. If not limited, than all classification problems can solved using the three layer network.

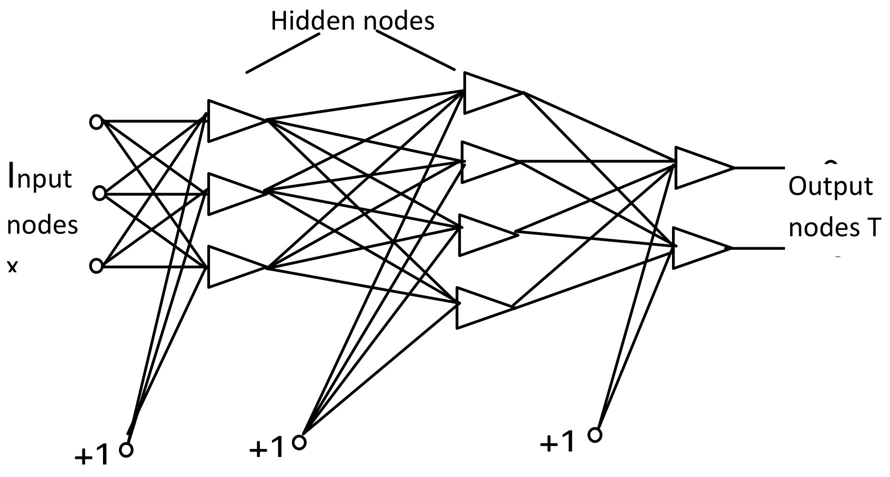

Figure 2.0: Feed-forward neural network.

Neurons in the first hidden layer create the separation lines for input clusters. Neurons in the second hidden layer perform AND operation, output neurons perform OR operation for each category. The linear separation property of neurons makes some problems especially difficult for neural networks, such as Ex-OR, parity computation for several bits, or to separable patterns laying on two neighboring spirals. The feed-forward neural network is also used for nonlinear transformation (mapping) of a multi-dimensional input variable into another multi-dimensional variable in the output. Any input –output mapping should be possible if neural network has enough neurons in hidden layers. Practically, it is not an easy task. Presently, there is no satisfactory method to define how many neurons should be used in hidden layers. Usually this is found by try and error method. In general, it is known that if more neurons are used, more complicated shapes can be mapped. On the other side networks with large number of neurons lose their ability for generalization, and it is more likely that such network will try to map noise supplied to the input also.

Gradient descent learning

Gradient descent is a first order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. Gradient descent is also known as steepest descent, or the method of steepest descent

A gradient descent based optimization algorithm such as back-propagation (BP) [6] can then be used to adjust connection weights in the ANN iteratively in order to minimize the error. The Gradient descent back-propagation algorithm [7] is a gradient descent method minimizing the mean square error between the actual and target output of multilayer perceptron. Assuming sigmoidal nonlinear function shown in fig2.1

$$f(net_i) = \frac{1}{1 - e^{-net}} \qquad 1$$

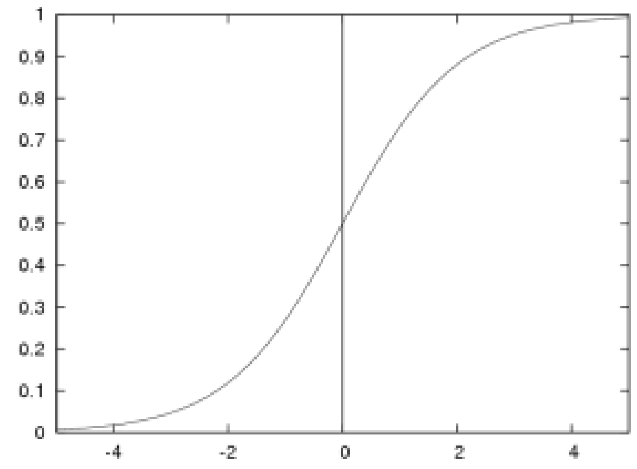

Figure 2.1: Sigmodal nonlinear function.

The Back-propagation [8], [9] networks tend to be slower to train than other types of networks and sometimes require thousands of epochs. When a reduced number of neurons are used the Error Back-propagation algorithm cannot converge to the required training error. The most common mistake is in order to speed up the training process and to reduce the training errors, the neural networks with larger number of neurons than required. Such networks would perform very poorly for new patterns nor used for training[10].

Gradient descent is relatively slow close to the minimum. BP has drawbacks due to its use of gradient descent [11, [12]. It 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. A detailed review of BP and other learning algorithms can be found in [13], [14], and [15].

Proposed Work

The validation of the gradient descent and proposed method are using XOR bench mark problem due to their 'simplicity' and not so high computation requirements.

The back-propagation algorithm

The back propagation algorithm consists of the following steps:

1. Initialization: Set all the weights and thresholds to small random values.
2. Presentation of input and desired (target) outputs Present the input vector X(1), X(2),….,X(N) and corresponding desired (target) response T(1),

T(2),….T(N), one pair at a time, where N is the total number of training patterns.

3. Calculation of actual outputs
4. Adaptation of weights and thresholds

To demonstrate back-propagation here we considered three layer, 7 neuron (2 input, 4 hidden, 1 output) feedforward network Since back-propagation and training requires thousands of steps the network would be initialized with random weights (The weights can be anything between -1 and 1),

The initial random weights for hidden layer are given by

0.8147	0.9134	0.2785	0.9649
0.9058	0.6324	0.5469	0.1576
0.1270	0.0975	0.9575	0.9706

The optimal trained hidden layer weights are

3.6970	1.8908	-7.1525	-6.9605
-7.3006	1.3526	3.4763	-7.0262
-1.0437	-1.5517	-0.9703	2.4692

The optimal Trained outer layer weights are

9.5114	-5.9792	8.7807	-10.3960

The fig 3.0 shows mean square error results of Back-propagation Artificial Neural Network (BP-ANN) in number of iterations. The table 3.0 gives the comparison of desired output and output of BP-ANN method. In this method the mean square error does not converge. So the accuracy is poor.

Table 3.0 Performance of xor using BP-ANN.

Input data		Desired output	Output by BP_ANN
0	0	0	0.0032
0	1	1.0000	0.9951
1	0	1.0000	0.9949
1	1	0	0.0076

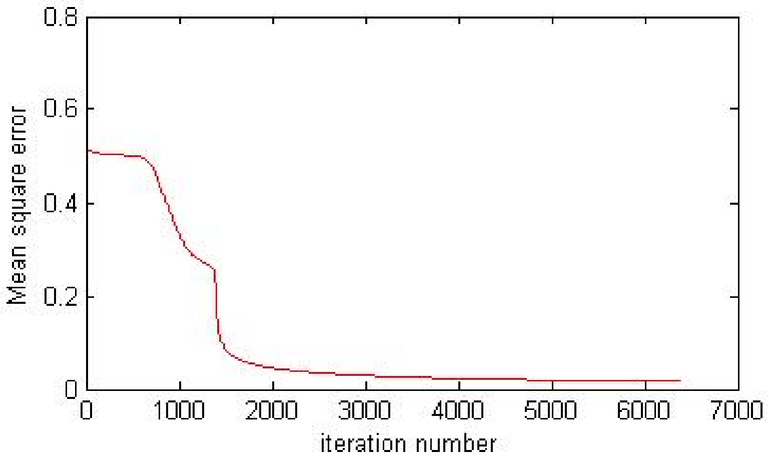

Figure 3.0: Performance of xor using BP-ANN.

Evolution of Connection weights

Weight training in ANN's is usually formulated as minimization of an error function, such as the mean square error between target and actual outputs averaged over all examples, by iteratively adjusting connection weights. Most training algorithms, such as BP [1], [2] given above and conjugate gradient algorithms [16], [17]–[19], are based on gradient descent. There have been some successful applications of BP in various areas [20], [21], [22].

One way to overcome gradient-descent-based training algorithms' shortcomings is to adopt EANN's, i.e., to formulate the training process as the evolution of connection weights in the environment determined by the architecture and the learning task. EA's can then be used effectively in the evolution to find a near-optimal set of connection weights globally without computing gradient information. The fitness of an ANN can be defined according to different needs. Two important factors which often appear in the fitness (or error) function are the error between target and actual outputs and the complexity of the ANN. Unlike the case in gradient-descent-based training algorithms, the fitness (or error) function does not have to be differentiable or even continuous since EA's do not depend on gradient information. Because EA's can treat large, complex, non-differentiable, and multimodal spaces, which are the typical case in the real world, considerable research and application has been conducted on the evolution of connection weights [23], [24], [25]

The aim of the proposed work is to find a near-optimal set of connection weights globally for an ANN with a fixed feed-forward architecture with three layer, 7 neuron (2 input, 4 hidden, 1 output) as shown in fig2.0 using Evolutionary Algorithm (EA). Various methods of encoding connection weights and different search operators used in EA. Comparisons between the evolutionary approach and conventional training algorithms, such as BP, have been made. In general, no single algorithm is an overall winner for all kinds of networks. The best training algorithm is problem dependent

Evolutionary algorithm:

1. Generate the initial population of μ individuals
2. Evaluate the fitness value for each individual of the population
3. Create λ offspring
4. Evaluate the fitness of each offspring
5. Sort offspring (or parents and offspring) and select μ best individuals to be parents of the next generation
6. Stop if stopping criterion is satisfied; Otherwise go to the step 3

Table 3.1: Performance of EA-ANN For XOR

Input data		Desired output	Output by GA_ANN
0	0	0	0.0000
0	1	1.0000	1.0000
1	0	1.0000	1.0000
1	1	0	0.0000

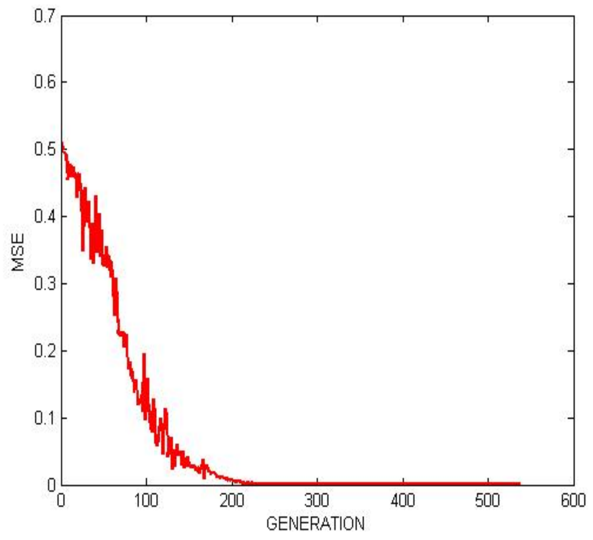

Figure 3.1: Best Chromosome Mean Square Error Plot for EA- ANN connection weights

The Fig.3.1 shows the mean square error for the best chromosome in the given population and table 3.1 shows the EX-OR truth table with high accuracy.

On comparison of two algorithms, the back-propagation takes 6684 iterations, total time taken is 9.4890 seconds and error minimization is 0.0178 to complete the task. But the evolutionary genetic algorithm takes 352 iterations, time taken is 9.4888 seconds, and error minimization is zero.

Conclusion

Evolutionary computation provides a powerful method for determination of weights, individual ANN architectures . A evolutionary algorithm is different form other classical search and optimization methods in a number of ways, It is a stochastic search and optimization procedure. A evolutionary genetic algorithm does not use gradient information, it works with a set of solutions instead of one solution in each iteration. The EA-ANN approach gave zero mean square error than the gradient-descent method datasets, and the results did not depend on the initial choice of weights. The objective of this research gave the increased performance of the network in terms of accuracy.

References

[1] X. Yao, "Evolution of connectionist networks," in *Preprints Int. Symp. AI, Reasoning & Creativity*, Queensland, Australia, Griffith Univ., pp. 49–52. 1991.

[2] --- "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

[3] ----, "Evolutionary artificial neural networks," *Int. J. Neural Syst.*, vol. 4, no. 3, pp. 203–222, 1993.

[4] ----, "The evolution of connectionist networks," in *Artificial Intelligence and Creativity*, T. Dartnall, Ed. Dordrecht, The Netherlands: Kluwer, pp. 233–243, 1994.

[5] ---- , "Evolutionary artificial neural networks," in *Encyclopedia of Computer Science and Technology*, vol. 33, A. Kent and J. G. Williams, Eds. New York: Marcel Dekker, pp. 137–170, 1995.

[6] G. E. Hinton, "Connectionist learning procedures," *Artificial Intell.*, vol. 40, no. 1–3, pp. 185–234, Sept. 1989

[7] Rumelhart D. E., Hinton G. E., Williams R. J. "Learning representations by back propagating errors", .Nature, 323, 533-536, 1986.

[8] Rumclhart D. E., Hinton G. E., Williams R. J.: " Learning errors" , Nature, Vol. 323, pp. 533-536, 1986.

[9] Wcrobs P. J,: " Back-propagation: Past and Future", Proc. Neural Networks, San Diego, CA, 1, 343-354, 1988.

[10] Wilamowski B. M,: " Neural Network Architectures and Learning Algorithms: How not to be Frustrated with Neural Networks, IEEE Industrial Electronics Magazine ", Vol. 3 No. 4, pp. 56-63, Dec. 2009.

[11] R. S. Sutton, "Two problems with back-propagation and other steepest-descent learning procedures for networks," in *Proc. 8th Annual Conf. Cognitive Science Society*. Hillsdale, NJ: Erlbaum, pp. 823–831, 1986

[12] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990

[13] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991.

[14] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Mag.*, vol. 10, pp. 8–39, Jan. 1993.

[15] Y. Chauvin and D. E. Rumelhart, Eds., *Back-propagation: Theory, Architectures, and Applications*. Hillsdale, NJ: Erlbaum, 1995.

[16] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991.

[17] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Mag.*, vol. 10, pp. 8–39, Jan. 1993.

[18] Y. Chauvin and D. E. Rumelhart, Eds., *Backpropagation: Theory, Architectures, and Applications*. Hillsdale, NJ: Erlbaum, 1995.

[19] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[20] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural Networks*, vol. 3, no. 1, pp. 33–43, 1990.

[21] S. Knerr, L. Personnaz, and G. Dreyfus, "Handwritten digit recognition by neural networks with single-layer training," *IEEE Trans. Neural Networks*, vol. 3, pp. 962–968, Nov. 1992.

[22] S. S. Fels and G. E. Hinton, "Glove-talk: A neural network interface between a data-glove and a speech synthesizer," *IEEE Trans. Neural Networks*, vol. 4, pp. 2–8, Jan. 1993.

[23] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990.

[24] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, pp. 116–121, 1989.

[25] P. Zhang, Y. Sankai, and M. Ohta, "Hybrid adaptive learning control of nonlinear system," in *Proc. 1995 American Control Conf. Part 4 (of 6)*, pp. 2744–2748, *1995.*