Least Square Method Based Evolutionary Neural Learning Algorithm

R. Ghosh and B. Verma
School of Information Technology
Griffith University, PMB 50, Gold Coast Mail Center
QLD 9726, Australia
r.ghosh@gu.edu.au, b.verma@gu.edu.au

Abstract

In this paper, we present a new idea of evolving weights for the Artificial Neural Networks (ANNs). We propose a novel hybrid learning approach for the training of a feed-forward ANN. The approach combines evolutionary algorithms with matrix solution methods such as Gram-Schmidt etc., to achieve optimum weights for hidden and output layers. Our hybrid method is to apply the evolutionary algorithm in the first layer and the least square method (LS) in the second layer of the ANN. A two-layer network is considered. The hidden layer weights are evolved using the evolutionary algorithm (EA). When a certain number of generation or error goal in terms of RMS/Class error is reached, the training stops. We start with a small number of hidden neurons, and then the number is increased gradually. We have applied our algorithm for XOR, 10-bit odd parity and handwritten segmented characters recognition problems. The implementation of the algorithm was done in MATLAB & C. Experiments show some promising results when compared with other evolutionary based algorithm only in terms of results in classification rate and time complexity.

Key words: Evolutionary algorithm, Least square method, Neural learning algorithm, Evolving neural networks

1 Introduction

Learning is the main essence of ANN. Learning can be considered as a weight-updating rule of the ANN. Most of the neural learning method strictly depends on the architecture of the ANN. The connection rules determine the topological structure of an ANN. Two most important factors in the weight updating rules are the ordering of the nodes and the connection mechanism of the degree of inputs to the ANN. The number of nodes determines the dimensionality of the problem. The response of the ANN can be realized as a function of the weights of an ANN. A decision hyper-plane can be

constructed by proper combinations of the weight vectors. These surface areas, which can be highly complex are the areas of concern for a learning algorithm, so that a decision hyper-plane can be constructed by proper combinations of the weight vectors.

There are many problems associated with the currently existing learning algorithms [1][2][3]. Some of the aspects with existing learning methods for MLP can be summarized as

- The convergence tends to be extremely slow
- Learning constants must be guessed heuristically
- Convergence to the global minimum is not guaranteed. [4]

EA based learning provide an alternative way to learn for the ANN. The task involves controlling the complexity by adjusting the number of weights of the ANN. The use of EA for ANN learning can be viewed as follows:

- Search for the optimal set of weights
- Search over topology space
- Search for optimal learning parameters
- A combination to search for all the above [5]

The fundamental work in this area was done by Holand, Rechenberg, Schwefel and Fogel during the 1970s [6]. Much of the research has focused on the training of feed forward networks [Fogel, Fogel, and Porto, 1990; Whitley, Starkweather, and Bogart, 1990] [7] [8]. Miller et al, reported that EA is a better candidate than other standard neural network techniques, because of the nature of the error surface[9] [10]. Those characteristics pointed out by miller are

- The architecture surface is infinitely large, hence unbounded for possible number of nodes and connections

- The surface is non-differentiable since changes in the number of nodes and connections are discrete
- The surface is complex and noisy since the mapping from the architecture to the performance is indirect, strongly epistasis, and dependent on the evaluation method used.
- The surface is deceptive since similar architectures may have quite different results.
- The surface is multi-modal since different architectures may have similar performance

To find a global solution in the search space, normally EA based learning takes a very long time to train the ANN. Hence, the time complexity of an EA based technique is a major problem. So to have a better time complexity, it is a good idea to combine the global search EA with the standard ANN learning methods, which perform some local search. Also, many proposals have been given by the researchers to apply such a hybrid algorithm, where the EA can find a good solution in terms of its weight and architecture of the ANN and then a normal local search procedure can be applied to find the final solution [11] [12] [13] [14].

Our aim is to investigate our novel approach and conduct a comparative study between the existing learning algorithms that uses EA for evolving the weights for the MLP, with our proposed hybrid learning that uses the combination of EA and least square method. In section 2 we discuss in details our hybrid learning method. In section 3 we discuss the results and finally in section 3, the conclusion and further research is given.

2 Proposed Learning algorithm

2.1 ANN architecture details

We consider two layers network architecture. The weights variable for each layer are found using a hybrid method, which uses the EA and a least square method. The evolutionary algorithm method is applied for the first layer weight and the least square methods are applied to find the weights for the output layer. We initialize the hidden layer weights with a uniform distribution with closed range interval $[-1, +1]$. Once the initialization for the hidden layer weights are done, we apply the least square method to obtain the weights for the output layer. After determining the weights for the output layer the EA is applied for the hidden layer to obtain an optimum weights. The algorithm stops after a certain number of generations or if a specific error goal is met.

2.2 New hybrid Algorithm

We conduct experiments using two different evolutionary algorithms, one (algorithm A) that uses both the genetic operators (crossover and mutation) and the other (algorithm B), which uses only mutation operator with a pair of real valued vectors. For comparison purpose we also train the same network, where the EA is applied for evolution of all the weights for hidden and output layer. We call the corresponding methods for our learning method A and B, as A' and B' respectively. The hybrid algorithm can be described as follows:

Step 1

Initialize the input range: All the inputs are mapped into a range of the open interval $(0,1)$.

Step 2

Start with a small number of hidden neurons: We start the training process using a small number of hidden neurons using a step incremental process. The initial number is chosen as small as possible.

Step 3

Initialize all the weights for the hidden layer: We initialize all the hidden layer weights using a uniform distribution of a closed interval range of $[-1, +1]$. We also encode for the genotype, which represents the weights for the hidden layer with those values for all the population strings. The genotype encoding also depends on the type of the EA to be used. For example, for algorithm A, we don't need any extra gene, where as for algorithm B, we need a pair of real valued vectors.

2.2.1 For Algorithm A

A sample genotype from the population pool for an n input, h hidden and m output neuron can be written as

$$|w_{11}w_{12}...w_{1n}w_{21}w_{22}...w_{2n}...w_{h1}w_{h2}...w_{hn}|$$

Where, $\text{range}(w)$ initially is set between the closed interval $[-1 +1]$

Also, as the output weights are found by the least square methods, those weights need not be initialized.

2.2.2 For Algorithm B

A sample genotype from the population pool for an n input, h hidden and m output neuron can be written as

$$|w_{11}\mu_{11}w_{12}\mu_{12}...w_{1n}\mu_{1n}w_{21}\mu_{21}w_{22}\mu_{22}...w_{2n}\mu_{2n}...w_{h1}\mu_{h1}w_{h2}\mu_{h2}...w_{hn}\mu_{hn}|$$

Where, $\text{range}(w)$ initially is set between the closed interval $[-1 +1]$

μ is are the variance vectors, each values of μ is initialized by a Gaussian distribution of mean 0 and standard deviation 1.

Step 4:

Compute the weights for the output layer using least square method: We compute the weights for the output layer using the least square method where the output of the hidden layer is computed as $f(\cdot)$ for the weighted sum of its input, where f is the sigmoid function.

The output of each of the hidden neuron can be found using the equations:

$$\left\{ h_i = f\left(\sum_{j=1}^n w_{ij} I_j\right) \right\}$$

Where $i = 1, 2, \dots, h$, I is the mapped input and

$$f = \frac{1}{1 + \exp^{-x}}$$

Where x is the output of the hidden neuron before the activation function.

After obtaining the corresponding weight gene from the genotype, as we use sigmoid activation function for the output also, we need to do the linearization, using the formula

$$netb_i = -\log\left(\frac{1 - net_i}{net_i}\right)$$

Where $i = 1, 2, \dots, m$, $netb$ is the output of the output neuron before the activation, and net is the output of the output neuron after the activation

We then require to solve the over determined equation

$$hid * weight = netb$$

Where hid is the output matrix from the hidden layer neurons and $weight$ is the weight matrix output neurons. We use least square method, which is based on the QR factorization technique to solve the equation for the weight matrix using the qr function

$$[Q, R] = qr(hid)$$

The function qr returns the orthogonal triangular decomposition of the hid matrix. It produces an upper triangular matrix R of the same dimension as hid and a unitary matrix Q so that $hid = Q * R$.

The solution matrix can be found from the R matrix using one step iterative process as

$$x = \frac{R}{R^T I(hid^T * netb)}$$

the error e can be calculated as

$$r = netb - hid * x$$

$$e = \frac{R}{R^T I(hid^T * r)}$$

The final value of solution for weight matrix can be then found as

$$weight = x + e$$

Step 5

Apply Evolutionary algorithm: We create an intermediate population from the current population using a selection operator. We use roulette wheel selection. The method creates a region of a wheel based on the fitness of a population string. All the population strings occupy the space in the wheel based on their rank in fitness. A uniform random number is then generated within a closed interval of $[0, 1]$. The value of the random number is used as a pointer and the particular population string that occupies the number is selected for the offspring generation. Once the intermediate population strings are generated, we randomly choose two parents from the pool of the intermediate population, and apply the genetic operators (crossover, mutation) to generate the offspring with some predefined probability patterns. We use fixed probability value for the genetic operators. Some preliminary results have shown that a crossover rate of $[0.7, 0.8]$ and mutation rate of $[0.1-0.2]$ provides the best results. We continue this process till such time the number of offspring population becomes same as the parent population.

Once the new population has been created, we find the fitness for all the population based on the weights of the hidden weights (obtained from the population string) and the output layer weights (obtained from the least square method). We also normalize the fitness value to force the population string to maintain a pre-selected range of fitness.

2.2.2.1 Intermediate population generation

$$netOutput = f(hid * weight)$$

Where f is the sigmoid function

$$RMSError = \sqrt{\frac{\sum_{i=1}^n (netOutput_i - net_i)^2}{n * h}}$$

$$popRMSError_i = norm(RMSError_i)$$

$norm$ function normalized the fitness of the individual, so the fitness of each individual population is forced to be within certain range.

2.2.2.2 Offspring generation Algorithm A

Generate a random number x from a Gaussian distribution of mean 0 and standard deviation 1.

If ($x < crossOverRate$)

Select two parents from the intermediate population

ApplyCrossOver

End If

Generate another random number y from the same distribution

If ($y < \text{mutationRate}$)

ApplyMutation

End If

The crossover method that is used for this algorithm is known as linear interpolation with two points technique. Let's consider two genes

$$w_1 w_2 \dots w_h$$

and

$$w'_1 w'_2 \dots w'_h$$

Two points are selected randomly, lets assume point1 and point2 , where $\text{point1} < \text{point2}$, and $\text{point1} > 1$, $\text{point2} < h$

The two child after the crossover operation will be

$$\frac{2w_1 + w'_1}{3} \frac{2w_2 + w'_2}{3} \dots \frac{2w_{\text{point1}} + w'_{\text{point1}}}{3} \frac{w_{\text{point1}} + 2w'_{\text{point1}}}{3} \dots \frac{w_{\text{point2}} + 2w'_{\text{point2}}}{3}$$

$$\frac{2w_{\text{point2}} + w'_{\text{point2}}}{3} \dots \frac{2w_h + w'_h}{3}$$

and

$$\frac{w_1 + 2w'_1}{3} \frac{w_2 + 2w'_2}{3} \dots \frac{w_{\text{point1}} + 2w'_{\text{point1}}}{3} \frac{2w_{\text{point1}} + w'_{\text{point1}}}{3} \dots \frac{2w_{\text{point2}} + w'_{\text{point2}}}{3}$$

$$\frac{w_{\text{point2}} + 2w'_{\text{point2}}}{3} \dots \frac{w_h + 2w'_h}{3}$$

For mutation, a small random value between 0.1 and 0.2, is added to all the weights. Let us assume a parent string

$$w_1 w_2 \dots w_h$$

After mutation the child string becomes

$$w_1 + \varepsilon \mid w_2 + \varepsilon \mid \dots \mid w_h + \varepsilon$$

Where ε is a small random number [0.1 0.2], generated using a uniform distribution.

Offspring generation Algorithm B

Each individual population gene (W_i, η_i), $i = 1, 2, \dots, \mu$ creates a single offspring (W'_i, η'_i) by : for $j = 1, 2, \dots, n$

$$\eta'_i(j) = \eta_i(j) \exp(\tau N(0,1) + \tau N_j(0,1))$$

$$W'_i(j) = W_i(j) + \eta'_i(j) N_j(0,1)$$

Where $W_i(j)$, $W'_i(j)$, $\eta_i(j)$, and $\eta'_i(j)$ denote the j th component of the vectors W_i , W'_i , η_i , and η'_i , respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean and variance of 0 and 1 respectively. $N_j(0,1)$ denotes that the random

number is generated a new for each value of j . The parameter τ and τ' are set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$

Step 6

Check the error goal: If any population string from the population pool meets the error goal criterion, we stop the algorithm. Otherwise, goto step 7.

Step 7

Increment the number of hidden neurons: Add one neuron in the hidden layer and goto step 3.

3 Results

We have implemented the algorithm in MATLAB and C. Our algorithm has been applied to XOR problem, 10 bits odd parity problem, and handwritten segmented characters data set. We first applied the evolutionary algorithm only to the ANN for all these data set to find the solution and then we compared the results with our proposed approach. The results show that in terms of time (number of generation required) is much less than that of the EA only. Also, in terms of RMS error, the result shows that an average result for the hybrid algorithm reaches the error goal with almost certainty. We only considered a very small number of hidden neurons to obtain results to compare the different learning algorithm. Result shows that even with a very small number of hidden neurons a considerable amount of error goal could be reached. Hence, keeping options open to improve our result further for future investigation of our technique.

Table 1: Classification Error Results

Data set	Algorithm	Class Error		
		Initial	Best	Final
XOR (Training pattern = 4)	A	0	0	0
	B	0	0	0
	A'	1	2	1
	B'	1	0	2
Parity (Training pattern = 1024)	A	477	477	497
	B	500	455	501
	A'	489	487	510
	B'	500	381	500
Hand written (Training pattern = 1000)	A	475	381	458
	B	810	10	10
	A'	880	880	965

Table 2: RMS Error Results

Data set	Algorithm	RMS Error		
		Initial	Best	Final
XOR (Training pattern = 4)	A	0.001	0.001	0.02
	B	0.022	0.02	0.02
	A'	0.225	0.01	0.225
	B'	0.225	0.205	0.22
Parity (Training pattern = 1024)	A	0.163	0.1633	0.16338
	B	0.162	0.162	0.162
	A'	0.165	0.165	0.165
	B'	0.162	0.162	0.162
Hand written (Training pattern = 1000)	A	0.051	0.01	0.01
	B	0.251	0.01	0.01
	A'	0.01	0.01	0.01
	B'	0.01	0.01	0.01

3.1 Analysis of the Results

3.1.1 Data Set 1

Both the new hybrid learning shows almost the same results for this data set. Two hidden neurons were found to be able to classify correctly the data set for all the four sets of algorithms. The convergence was almost certain for all the 4 algorithms, however the result shows that convergence for algorithm A' and B' were slightly better than the proposed algorithm A and B, though the minimum error in case of the proposed algorithm were found much earlier in terms of number of generations. Also, the time taken by the new algorithms A and B were much less. On an average the algorithm A and B were able to find the goal within one third time than the time taken by algorithm A' and B'.

3.2 Data Set 2

Result shows that the algorithm A achieves a better classification rate (58%) with only 5 hidden neurons, than algorithm A with classification rate (56%). In both cases the best classification result was found using the least number of neurons out of three. Algorithm B, once gave a classification result of 99%, though this result was an extreme result found after the algorithm was run for several times. Algorithm A' gives a correct classification of 54%, where as algorithm B' gives a correct classification of 63%. The average time taken by algorithm A and B were 10 times faster than the time taken by algorithm A' and B'. The convergence property was almost the same for all the four algorithms.

3.2.1 Data Set 3

Result shows that the algorithm A achieves a better classification rate (64%) with only 5 hidden neurons, than algorithm B with classification rate (34%). In both the cases the best classification result was found using the least number of neurons out of three. But twice algorithm B gave a classification result of 99%, though this result was an extreme result found after the algorithm was run for several times. Algorithm A' gives a correct classification of 20% same as algorithm B'. The average time taken by algorithm A and B were 10 times faster than the time taken by algorithm A' and B'. The convergence property was better for algorithm A and B than algorithm A' and B'.

4 Conclusion and Further research

The new algorithm shows some promising results in terms of the classification rate and time complexity when compared with the existing EA based techniques to train an ANN. In terms of convergence property the existing EA based learning shows better results, as far as steady decrease in error is concerned. But in terms of the probability the solution in case of hybrid learning was almost certain when compared to the existing EA based techniques. There are many scopes to improve the existing learning algorithm for both the EA methods and the matrix solution methods. The result could be improved more using more hidden neurons, but the main aim at this stage was to compare the results between various algorithm given the time constraint. Also, a larger benchmark data set need to be used for further verification. Future research will also take into consideration the EA method to find the number of hidden neurons in the first layer.

5 Reference

- [1] P. Whittle, "Prediction and regularization by linear least square methods", Van Nostrand, Princeton, N.J., 1963
- [2] S. D. Goggin, K.E. Gustafson, and K. M. Johnson, "An asymptotic singular value decomposition analysis of nonlinear multilayer neural networks," International Joint Conference on Neural Networks, pp I-785-I-789, 1991.
- [3] S. A. Burton, "A matrix method for optimizing a neural network," Neural comput. Vol 3, no 3.
- [4] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, "What size neural network gives optimal generalization? Convergence properties of backpropagation", UMIACS-TR-96-22
- [5] V.Petridis, S.Kazarlis, A.Papaikonomu and A.Filelis. "A hybrid genetic algorithm for

-
- training neural network". Artificial Neural Networks, 2, 953-956, 1992
- [6] I. Rechenberg, "Cybernetic solution path of an experimental problem.", Royal Aircraft Establishment, Library translation no. 1122, Farnborough, Hants, U.K, Aug, 1965
 - [7] Whitley, D., Starkweather, T., & BoEArt, C." Genetic algorithms and neural networks - optimizing connections and connectivity". Parallel Computing, 14, 347-361, 1990.
 - [8] Montana, D. & Davis, L. "Training feedforward neural networks using genetic algorithms". Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89, 1, 1989.
 - [9] M. Frean, "The upstart algorithm: a method for constructing and training feedforward neural networks," Neural computation, vol 2, 1990
 - [10] A. Roy, L.S. Kim, and S. Mukhopadhyay, "A polynomial time algorithm for the construction and training of a class of multiplayer perceptrons," Neural networks, vol 6, 1993.
 - [11] D.R. Hush and B.G. Horne, "Progress in supervised neural networks," IEEE signal processing MaEAzine, vol. 10, no. 1, pp. 8-39, Jan 1993
 - [12] M.F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," Neural networks, vol. 6, pp. 525-523, June 1993.
 - [13] R.K. Belew, J. McInerney, and N.N. Schraudolph, "Evolving networks: using genetic algorithm, with connectionist learning," Tech. Rep. #CS90-174 (Revised), Computer Science & Engr. Dept. (C-014), Univ. of California at San Diego, La Jolla, CA 92093, USA, Feb 1991.
 - [14] A.P. Topchy and O.A. Lebedko, "Neural network training by means of cooperative evolutionary search," Nuclear Instruments & Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 389, no. 1-2, pp. 240-241, 1997.