# A Novel Evolutionary Neural Learning Algorithm

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

## ABSTRACT

**In this paper, we present a novel genetic algorithm and least square (GALS) based hybrid learning approach for the training of an artificial neural network (ANN). The approach combines evolutionary algorithms with matrix solution methods such as Gram-Schmidt, SVD, etc., to adjust weights for hidden and output layers. Our hybrid method (GALS) incorporates the evolutionary algorithm (EA) in the first layer and the least square method (LS) in the second layer of the ANN. In the proposed approach, a two-layer network is considered, the hidden layer weights are evolved using an evolutionary algorithm and the output layer weights are calculated using a linear least square method. When a certain number of generation or error goal in terms of RMS error is reached, the training is stopped. We start training with a small number of hidden neurons and then the number is increased gradually in an incremental process. The proposed algorithm was implemented and many experiments were conducted on benchmark data sets such as XOR, 10-bit odd parity, handwritten segmented characters recognition, breast cancer diagnosis and heart disease data. The experimental results showed very promising results when compared with other existing evolutionary and error back propagation (EBP) algorithm in classification rate and time complexity.**

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

## I INTRODUCTION

One of the most popular weight-updating rules or learning (training) algorithms is Error Back Propagation (EBP). However, most of the EBP based neural learning algorithms strictly depends on the architecture of the ANN and there are many problems associated with the currently existing algorithms based on EBP and its variations [1-4]. Some of the problems with existing EBP type learning algorithms for ANNs can be summarized as below:

- The convergence tends to be extremely slow, sometimes it takes many days or weeks even on supercomputer.
- The solution is not guaranteed even after many days or weeks because of local minima & paralysis problems.
- Learning parameters such as number of hidden nodes, iterations, etc. must be guessed heuristically or by a trial and error method.
- Convergence to the global minimum is not guaranteed.

There were a number of hybrid techniques [5-20] proposed to improve EBP type learning algorithms by using least square methods (LSM), evolutionary algorithms (EA), etc. EA & LSM based learning provide an alternative way to train an ANN. The learning algorithms using EA is based on primary work in genetic algorithms by Holand, Rechenberg, Schwefel and Fogel during the 1970s[1]. Much of the research has focused on the training of feed forward networks [Fogel, Fogel, and Porto, 1990; Whitley, Starkweather, and Bogart, 1990][1-2]. Miller et al, reported that EA is a better candidate than other standard neural network techniques, because of the nature of the error surface[1].

The main aim of the research presented in this paper was to investigate a novel hybrid learning approach and conduct a comparative study between the existing learning algorithms that uses EA for evolving the weights for the MLP, with the proposed hybrid learning that uses the combination of EA and a least square method.

## II ORGANISATION OF THE PAPER

In section 3 we present in detail our evolutionary learning algorithm. In section 4 we discuss the experimental results and finally in section 5, the conclusion is given.

## III PROPOSED LEARNING ALGORITHM

We consider two layers feed-forward neural network architecture. The evolutionary algorithm is applied for the first layer weight and the least square method is applied to find the weights for the output layer. The evolutionary algorithm that is incorporated for the hidden layer is the key for the convergence property of the proposed algorithm. In our original GALS, we study the results using the standard genetic algorithm operations. We, then modify the standard GALS using the concept of evolutionary algorithm, where only mutation operation is considered. GALS is described below in detail using steps 1-7:

Step 1
*Initialize the input range:* All the inputs are mapped into a range of the open interval (0,1). The method of normalization is based on calculating the mean and standard deviation for each element column and use these to perform additional scaling if required.

Step 2
*Start with a small number of hidden neurons:* We start the training process using a small number of hidden neurons and a step incremental process.

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]. The genotype encoding also depends on the type of the EA to be used.

*III.I.I Standard GALS*

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

$$\left| w_{11} w_{12} \ldots w_{1n} w_{21} w_{22} \ldots w_{2n} \ldots w_{h1} w_{h2} \ldots w_{hn} \right|$$

Where, range(w) initially is set between the closed interval [-1 +1]

*III.I.II Modified GALS*
A sample genotype from the population pool for an n input, h hidden and m output neuron can be written as

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

Where, range(w) initially is set between the closed interval [-1 +1]
$\mu$ are the variance vectors, each values of $\mu$ 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(·) for the weighted sum of its input, where f is a sigmoid function.

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

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

Where i = 1,2,...,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,...,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 mathematical details of the algorithm are as follows:

We can solve Ax=b by calculating the QR factorization of A and solving first Qy=b (hence y = $Q^Tb$) and then Rx=y (a triangular system).

Let

$$u \in R^m \setminus \{0\}$$

$$H = I - 2\frac{uu^T}{\|u\|^2}$$

is called a Householder transformation or a Householder reflection. Since Hu = -u, Hv = v if $u^Tv = 0$, this transformation reflects any vector x ∈ $R^n$ in the (n-1) dimensional hyper plane spanned by the vectors orthogonal to u. Each such matrix H is symmetric and orthogonal. The later follows because reflection leaves the Euclidian distance invariant, or by direct calculation

$$\left(I-2\frac{uu^T}{\|u\|^2}\right)^T\left(I-2\frac{uu^T}{\|u\|^2}\right)=\left(I-2\frac{uu^T}{\|u\|^2}\right)^2$$

$=$

$$I-4\frac{uu^T}{\|u\|^2}+4\frac{u(u^Tu)u^T}{\|u\|^4}=I$$

Let a $\in$ $R^m$ be the first nonzero column of A. We wish to choose u $\in$ $R^m$ and, in addition, we normalize u so that $2u^Ta=\|u\|^2$ (and $a\neq 0$)

Therefore $u_i$ = $a_i$,i = 2,...,m and the normalisation implies that

$$2u_1a_1+2\sum_{i=2}^{m}a_i^2=u_i^2+\sum_{i=2}^{m}a_i^2 =>$$

$$u_i^2-2u_1a_1+a_i^2-\sum_{i=1}^{m}a_i^2=0 =>$$

$$u_1=a_1\pm\|a\|$$

It is usual to let the sign be the same as the sign of $a_1$, since $\|u\|<<1$ might lead to a division by a tiny number, hence to numerical difficulties. For large m we do not execute explicit matrix multiplication. Instead, to calculate

$$\left(I-2\frac{uu^T}{\|u\|^2}\right)A=A-2\frac{u(u^TA)}{\|u\|^2}$$

We first evaluate $w^T = u^TA$, subsequently forming

$$A-\frac{2}{\|u\|^2}uw^T$$

Supposing that the Householder transformation has been applied k-1 times, so that the first k-1 columns of the resultant matrix A have an upper triangular form.

We process columns A in sequence, in each stage pre multiplying a current A by the requisite Householder transformation. The end result is an upper triangular matrix R. To determine Q, we set $\Omega$ = I initially, and for each successive reflection, we replace $\Omega$ by

$$\left(I-2\frac{uu^T}{\|u\|^2}\right)\Omega=\Omega-\frac{2}{\|u\|^2}u\left(u^T\Omega\right)$$

As in the case of given rotation, by the end of the computation, Q = $\Omega^T$. However if we require just vector c = $Q^Tb$, rather than matrix Q, then we can set initially c = b and in each stage we replace c by

$$\left(I-2\frac{uu^T}{\|u\|^2}\right)c=c-\frac{2u^Tc}{\|u\|^2}u$$

If A is dense, it is in general more convenient to use Householder reflections. Given rotations come into their own, however when A has many leading zeros in its rows. In an extreme case, if an n X n matrix A consists of zeros underneath the first sub diagonal, they can be rotated away in just n-1 rotations at the cost of $0(n^2)$ operations.

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

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

the error e can be calculated as

$$r=netb-hid*x$$

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

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.

In case of standard GALS, we call the function to calculate the weights of the output layer, based on a constant population string of the hidden layer. Once a fixed population string was determined, we could then call the least square method. In case of modified GALS, we define two different fitness functions. One fitness function was called initially after randomizing the population for the hidden layers, and then giving the best population index finding the fitness for all the population and then calling the least square method for all of them.

*III.I.II.1 Intermediate population generation*

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

Where f is the sigmoid function

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

$$popRMSError_i = norm(RMSError_i)$$

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

*III.I.II.2 Offspring generation for standard GALS*

```
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 ($y$ < 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

$$w_1 w_2 .. w_h$$

$$and$$

$$w_1^{/} w_2^{/} ... w_h^{/}$$

Two points are selected randomly, lets assume *point1* and *point2*, where *point1<point2*, and *point1*>1, *point2<h*

The two childs after the crossover operation will be

$$\frac{2w_1+w_1^{/}}{3}\frac{2w_2+w_2^{/}}{3}\ldots\frac{2w_{point1}+w_{point1}^{/}}{3}\frac{w_{point1+1}+2w_{point1+1}^{/}}{3}\ldots\frac{w_{point2-1}+2w_{point2-1}^{/}}{3}$$
$$\frac{2w_{point2}+w_{point2}^{/}}{3}\ldots\frac{2w_h+w_h^{/}}{3}$$

$$and$$

$$\frac{w_1+2w_1^{/}}{3}\frac{w_2+2w_2^{/}}{3}\ldots\frac{w_{point1}+2w_{point1}^{/}}{3}\frac{2w_{point1+1}+w_{point1+1}^{/}}{3}\ldots\frac{2w_{point2-1}+w_{point2-1}^{/}}{3}$$
$$\frac{w_{point2}+2w_{point2}^{/}}{3}\ldots\frac{w_h+2w_h^{/}}{3}$$

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

$$w_1 w_2 .. w_h$$

After mutation the child string becomes

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

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

*Offspring generation for modified GALS*

Each individual population gene (Wi, ηi) , I = 1, 2, …, μ creates a single offspring (Wi$^{/}$, ηi$^{/}$) by : for j = 1, 2, …, 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 jth component of the vectors Wi, Wi$^{/}$, $\eta_i$, and $\eta_i^{/}$, respectively. N(0,1) denotes a normally distributed one-dimensional random number with mean and variance of 0 and 1 respectively. Nj(0,1) denotes that the random number is generated a new for each value of j. The parameter τ and τ$^{/}$ are set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$

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 more neuron in the hidden layer and goto step 3.

## IV EXPERIMENTAL RESULTS AND ANALYSIS

We have implemented the proposed algorithm in MATLAB and C. We have tested the proposed algorithm on many benchmark data sets such as XOR, 10 bits odd parity, handwritten characters, breast cancer and heart disease. We compare the GALS results with other approaches like existing EBP and GA based algorithms.

TABLE 1 RESULTS FOR EBP[1]

| Data Set | Class Error | | Time[2] |
|---|---|---|---|
| | Train | Test | |
| XOR data set | 1 (25%) | X | 10 mins |
| 10 bit odd parity data set | 360 (35.15%) | X | 63 mins |
| Handwriting data set | 110 (36.6%) | 26 (52%) | 61 mins |
| Breast cancer data set | 146 (36.5%) | 134 (44.8%) | 57 mins |
| Heart disease data set | 100 (40%) | 27 (50.9%) | 52 mins |

TABLE 2 RESULTS FOR MODIFIED GALS[3]

| Data set | Population | Hidden Neuron | Time | Class Error | |
|---|---|---|---|---|---|
| | | | | Train | Test |
| XOR data set | 10 | 2 | 250 secs | 1 (25%) | X |
| | 20 | 2 | 262 secs | 0 (0%) | X |
| 10 bit odd parity data set | 10 | 10 | 62 mins | 300(29.3%) | X |
| | | 20 | 64 mins | 290(28.3%) | X |
| | 20 | 10 | 71 mins | 287(28%) | X |
| | | 20 | 72 mins | 286(27.9%) | X |
| Hand wri-ting char-acter data set[4] | 10 | 10 | 40 mins | 135(45%) | 23(46%) |
| | | 20 | 41 mins | 130(43.3%) | 21(42%) |
| | | 30 | 43 mins | 126(42%) | 22(44%) |
| | 20 | 10 | 49 mins | 131(43.6%) | 24(48%) |
| | | 20 | 51 mins | 125(41.6%) | 21(42%) |
| | | 30 | 53 mins | 121(40.3%) | 19(38%) |
| | | 20 | 54 mins | 121(32.2%) | 107(35.7%) |
| | | 30 | 55 mins | 110(27.5%) | 100(33.4%) |
| | 20 | 10 | 62 mins | 117(29.2%) | 104(34.7%) |
| | | 20 | 65 mins | 110(27.5%) | 97(32.4% |
| | | 30 | 66 mins | 103(25.7%) | 91(30.4%) |
| Heart dis-ease data set[5] | 10 | 10 | 41 mins | 68(27.2%) | 23(42.3%) |
| | | 20 | 43 mins | 65(26.0%) | 22(41.5%) |
| | | 30 | 45 mins | 63(25.2%) | 20(37.7%) |
| | 20 | 10 | 47 mins | 65 (26.0%) | 21(39.6%) |
| | | 20 | 48 mins | 64(25.6%) | 19(35.8%) |
| | | 30 | 50 mins | 61(24.4 %) | 18(33.9%) |
| Breast cancer data set[6] | 10 | 10 | 52 mins | 130(32.5%) | 118(39%) |
| | | 20 | 54 mins | 121(32.2%) | 107(35.7%) |
| | | 30 | 55 mins | 110(27.5%) | 100(33.4%) |
| | 20 | 10 | 62 mins | 117(29.2%) | 104(34.7%) |
| | | 20 | 65 mins | 110(27.5%) | 97(32.4%) |
| | | 30 | 66 mins | 103(25.7%) | 91(30.43%) |

[1] In case of EBP the number of epochs were varied from 500 to 3000, the maximum number of hidden neurons were 10, learning rate 0.2 and momentum 0.7

[2] The time complexity of EBP was calculated after adding all the runs to find the best solution, which included changing the learning parameters and number of hidden neurons.

[3] Results are only given for modified GALS, which was better than the standard GALS.

[4] For handwriting data set the number of training pattern was 300 and testing 50. Number of input data column was 100 and output column 29 (26 characters and 3 rubbish characters).

[5] For heart disease data set the number of training pattern was 250 and testing 53. Number of input data column was 14 and output column 1.

[6] For breast cancer data set the number of training pattern was 400 and testing was 299. The number of input column was 10 and output 1.

## V CONCLUSION AND FURTHER RESEARCH

The proposed evolutionary neural learning algorithm showed some promising results in terms of the classification rate and time complexity when compared with the existing EA and EBP based techniques to train an ANN. As shown in Tables 1 and 2, the proposed evolutionary learning algorithm GALS performed better in terms of classification error and took less training time than EBP. Also the experiments showed that the probability of getting guaranteed solution is much higher for GALS than the existing EA and EBP based techniques. There are many scopes to improve the proposed learning algorithm for both the hidden and output layers. In our future research, the automatic selection of hidden neurons as described in section III, will be implemented and investigated.

## References

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.ing, vol . 3, no 3, 1994.
4. S. Lawrence, L. Giles, and A. C. Tsoi, "What size neural network gives optimal generalization? Convergence properties of backpropagatioin", UMIACS-TR-96-22.
5. D. Whitley and T. Starkweather" Genetic algorithms and neural networks - optimizing connections and connectivity". Parallel Computing, 14, 347-361, 1990.
6. D. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms". Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89, 1, 1989.
7. M. Frean, "The upstart algorithm: a method for constructing and training feedforward neural networks," Neural computation, vol 2, 1990.
8. 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.
9. B.K. Verma, "Fast Training of Multilayer Perceptrons (MLPs)", IEEE Transactions on Neural Networks, vol. 8, no. 6, pp. 1314-1321, 1997.
10. X. Yao, "Evolving Artificial Neural Networks, Proceedings of the IEEE", vol. 87, no. 9, 1423-1447, 1999.
11. X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster", IEEE Transactions on Evolutionary Computation, 3(2):82-102, July 1999.
12. A.J.F. Rooij, L. C. Jain and R. P. Johnson, Neural Network Training Using Genetic Algorithms, World Scientific Publishing Company, 1996.
13. H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in Proc. of the Eighth Nat'l Conf. on AI (AAAI-90), pp. 789-795, MIT Press, Cambridge, MA, 1990.
14. H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park, "Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates," Neurocomputing, vol. 11, no. 1, pp. 101-106, 1996.
15. J. C. F. Pujol and R. Poli, "Evolving the topology and the weights of neural networks using a dual representation," Applied Intelligence, vol. 8, no. 1, pp. 73-84, 1998.
16. K. S. Tang, C. Y. Chan, K. F. Man, and S. Kwong, "Genetic structure for NN topology and weights optimization," in Proceedings of the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALE-SIA'95), (Stevenage, England), pp. 250-255, IEE Conference Publication 414, 1995.
17. M. Srinivas and L. M. Patnaik, "Learning neural network weights using genetic algorithms-improving performance by search-space reduction," in Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Singapore), vol. 3, pp. 2331-2336, IEEE Press, New York, NY, 1991.
18. P. J. Angeline, G. M. Sauders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," IEEE Trans. on Neural Networks, vol. 5, no. 1, pp. 54-65, 1994.
19. R. E. Smith and I. H. B. Cribbs, "Combined biological paradigms: a neural, genetics-based autonomous systems strategy," Robotics and Autonomous Systems, vol. 22, no. 1, pp. 65- 74, 1997.
20. R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation," Decision Support Systems, vol. 22, no. 2, pp. 171-185, 1998.