A hybrid neural learning algorithm combining evolutionary Algorithm with discrete gradient method

Moumita Ghosh, Adil Bagirov, Ranadhir Ghosh, John Yearwood
School of Information Technology and Mathematical Sciences, University of Ballarat
PO Box 663, Victoria
Australia
{m.ghosh, a.bagirov, r.ghosh, j.yearwood} @ballarat.edu.au

Abstract- **In this paper we investigate a hybrid model based on the Discrete Gradient method and an evolutionary strategy for determining the weights in a feed forward artificial neural network. The Discrete Gradient method has the advantage of being able to jump over many local minima and find very deep local minima. However earlier research has shown that a good starting point for the discrete gradient method can improve the quality of the solution point. Evolutionary algorithms are very suitable for global optimisation problems. Nevertheless they are cursed with longer training time often unsuitable for real world applications. For optimisation problems such as weight optimisation for ANNs in real world applications the dimensions are large and time complexity is critical. Hence the idea of a hybrid model can be a suitable option. In this paper we propose a hybrid model combining an evolutionary strategy with the discrete gradient method to obtain an optimal solution much quicker. Comparative results on a range of standard datasets are provided with other hybrid neural learning models such as an evolutionary strategy combined with least square based methods. The comparative results are also given based on the evolutionary based search and discrete gradient. The results confirm that, in general improved weight determination is delivered and the memory complexity is improved.**

I. INTRODUCTION

A learning algorithm is at the heart of a neural network based system. Over the past decade, a number of learning algorithms have been developed. However, in most cases learning or training of a neural network is based on a trial and error method. There are many fundamental problems such as a long and uncertain training process, selection of network topology and parameters that still remain unsolved. Learning can be considered as a weight-updating rule of the ANN.

Error Back Propagation (EBP) is probably the most cited learning algorithm in the field of Artificial Neural Networks (ANNs) [9]. Rumelhart et al [10] developed the back propagation learning for the Multi Layer Perceptron. Back-propagation is based on the *gradient descent* minimization method. The ANN is presented with an input pattern, for which an output pattern is generated. Then, the error between desired and actual output can be determined, and passed backwards through the ANN. Based on these errors, weight adaptations are calculated, and errors are passed to a previous layer, continuing until the first layer is reached. The error is thus propagated back through the ANN. Most of the calculus-based ANN algorithms depend on the gradient information of the error surface, which may not always be available or expensive to find. Also the algorithm may very easily be trapped in a local minimum [3] – [4].

One of the alternative learning techniques that attracted research is the genetic algorithm. Genetic algorithms are a stochastic search method introduced in the 1970s in the United States by John Holland [11] and in Germany by Ingo Rechenberg [12]. Much of the research however has focused on the training of feed forward networks [13] - [14]. Just as neurobiology is the inspiration for artificial neural networks, genetics and natural selection are the inspiration of the genetic algorithm. It is based on a Darwinian type `survival of the fittest' strategy. An advantage of using GAs for training neural networks is that they may be used for networks with arbitrary topologies. Also, GAs do not rely on calculating the gradient of the cost function. Cost functions need to be calculated to determine their fitness. Because of the stochastic nature of this algorithm the learning process can reach an optimal solution with much higher probability than many standard neural based techniques, which are based on the gradient information of the error surface.

One of the problems though, with this global search based technique is the time complexity of the algorithm. For a very large application size, a very powerful computation facility is required to solve the problem. Hence there was a further need for an improvement of this approach in terms of the time complexity (and to some extent the quality of solution) by fine tuning the search within the local neighborhood area of the global solution obtained by the genetic algorithm. This suggests that a hybrid of the GA and some other fine tuning algorithm could be advantageous. Most of the hybrid algorithms developed for ANNs have used GA and some kind of a local search method. Amongst the local search techniques, EBP has been used most extensively. Earlier research in this area had shown that hybrid training was successful [15] –[16]. There were a number of researchers who used GA and EBP hybrids and reported an improvement of the algorithm over traditional GA or EBP [17] – [18]. Some recent work also suggested an improvement for hybrid algorithms by running several parallel combinations of global and local search [19] – [21].

Earlier work by Ghosh and Verma [9], suggested an alternative learning methodology, which uses a hybrid technique by using evolutionary learning for the hidden layer weights and least square based solution method for the output layer weights. The proposed algorithm solved the problems of time complexity of the evolutionary algorithm by combining a fast searching methodology using a least squares technique. However the memory complexity was quite high. The order of memory complexity on average could be 4-5 times higher

than EBP. The high memory complexity order was due to the use of extensive matrix operations for the least squares method, which has high memory demands for solving the linear equations to find output layer weights. The other problem with the method was producing large weights for the output layers. Such large weight modification tends to destroy previously acquired knowledge and likely thus decrease the generalizing ability of the neural network.

Derivative free methods seem to be the best option to deal with this kind of problem with a large number of variables (weights). Such methods can overcome stationary points, which are not local minima and some shallow local minima. Two widely used derivative free methods – the Powell method and the Nelder-Mead Simplex method are effective when the objective function is smooth and the number of variables is less than twenty. However in many problems the number of variables is much larger than twenty and sometimes the objective function is non-smooth. Another popular derivative free method is the discrete gradient method.

Bagirov et al in 2004 [22] have applied *discrete gradient* methods in generating the neural network weights. The discrete gradient is a finite difference estimate to a subgradient. Unlike many other finite difference estimates to subgradient, the discrete gradient is defined with respect to a given direction, which allows a good approximation for the quasidifferential. The algorithm calculates discrete gradients step by step, and after a finite number of iterations either the descent direction is calculated or it is found that the current point is an approximate stationary point. In the Discrete gradient method Armijo's algorithm is used for a line search. Hence at a given approximation, the method calculates the descent direction by calculating the discrete gradients step by step, and improving the approximation of the Demayibv-Rubinov quasidifferential. Once the descent direction is calculated, Armijo's algorithm is used for line search. The local minima is chosen as the next approximation. Hence the Discrete Gradient method jumps over many local minima and finds very deep local minima. However earlier research has shown that a good starting point for the discrete gradient method can improve the quality of the solution point. In this paper we combine an Evolutionary algorithm with the discrete gradient method to find the weights in the neural network. The evolutionary algorithm generates a near optimal solution after several generations. That near optimal solution is passed through the discrete gradient method as a starting point. The discrete gradient method generates the final solution.

II. METHODOLOGY

A. Discrete gradient method

The discrete gradient method is a *bundle method* where the sub-gradients are replaced by their approximations. Detailed description of this method can be found in [5] – [7]. The Discrete gradient of a function *f* at a point *x* is defined with respect to a given direction (*g*) and is calculated using a step (λ) along that direction. The coordinates of the discrete gradient are defined as finite difference estimates to a gradient in some neighbourhood of the point $x+\lambda g$. The ith coordinate of the discrete gradient is defined so as to approximate a sub gradient of the function *f*. Thus discrete gradient contains some information about the behaviour of the function f in some region around the point *x*.

The next step is to compute the descent direction. We take any direction and calculate the first discrete gradient. Then we calculate the least distance between the convex hull of the discrete gradients and the origin. This problem is reduced to a quadratic programming problem and can be effectively solved by Wolf's terminating algorithm. If this distance is less than some tolerance $\partial > 0$, the algorithm stops and we consider that point as an approximated stationary point. Otherwise a search direction is calculated. If this direction is a descent direction, the algorithm terminates, else we calculate a new discrete gradient with respect to this direction to improve the approximation of the set of generalised gradients. Since the discrete gradient contains some information about the behaviour of the function in some regions around the point *x*, this algorithm can find descent directions in stationery points, which are not local minima.

Supervised learning of a neural network is an optimisation problem [1] – [2], that involves minimising the error function given some set of training data. Hence the neural network weights can be determined using the discrete gradient method (DG) with the sum of squared error function. The sum of squared error function is defined as

$$\min E(w_0, w_h) = \sum_{p=1}^{pat}\sum_{k=1}^{K}\left(O_{pk} - y_{pk}\right)^2$$

where $O_p = act\left(w_o^T hid_p\right)$ and $hid_p = act\left(w_h^T x_p\right)$

Here *act* denotes the activation function, W_O denotes the matrix of output layer weights, W_k denote hidden layer weights, x_p denotes the pth input pattern, y_p the pth output pattern and hid_p denotes the hidden layer output for pth pattern and O_p denotes the output layer output for pth pattern. pat denotes the total number of training pattern exists.

Let $W = (W_h, W_O)$. The Discrete gradient method applied in neural network learning is as follows

- **Step1**: Choose any starting point W^0 at $k = 0$.
- **Step2**: Set $s = 0$ and $W_s^k = W^k$.
- **Step3**: Calculate the descent direction at $W = W_s^k$ and $\partial = \partial^k$.
- **Step 4**: We calculate the least distance between the convex hull of the discrete gradients and the origin.

$$\left\|v_s^k\right\| = \min\left\{\|v\| : v \in \overline{D}_m\left(W_s^k\right)\right\}$$

- **Step 5**: Calculate the search direction (g_s^k).

$$g_s^k = -\left\|v_s^k\right\|^{-1} v_s^k$$

- **Step 6**: If $\left\|v_s^k\right\| \le \partial_k$ then set $W^{k+1} = W_s^k$, $k = k+1$ and go to Step 2.
- **Step 7**: Construct the following iteration $W^{k+1} = W_s^k + \sigma_s g_s^k$, where σ_s is defined as follows

$$\sigma_s = \arg\max\left\{\sigma \ge 0 : f\left(W_s^k + \sigma g_s^k\right) - f\left(W_s^k\right) \le c\sigma\left\|v_s^k\right\|\right\}$$

- **Step 8**: Set $s = s+1$. Go to step 3.

B. Evolutionary Algorithm

Evolutionary algorithms (EAs) are search methods that take their inspiration from natural selection and survival of the fittest in the biological world. EAs differ from more traditional optimization techniques in that they involve a search from a "population" of solutions, not from a single point. Each iteration of an EA involves a competitive selection that rejects the poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen.

Let $W = (W_h, W_O)$ be an n dimensional solution vector and σ be the corresponding step size. Let m be the number of the population in a generation where each population is the pair (W_e, σ_e).

In the first generation m populations are generated randomly. In the subsequent generations the population set is created by selection and mutation. The Evolutionary algorithm is as follows

- **Step1**: Randomly initialize *m* population vector.
- **Step2**: The parents are mutated as follows

  for *j* = 1, 2, …, *n*

$$\sigma_i'(j) = \sigma_i(j)\exp\left(\tau' N(0,1) + \tau N_j(0,1)\right)$$

$$W_i'(j) = W_i(j) + S\sigma_i'(j)N_j(0,1)$$

  where the values of τ' and τ as follows

$$\tau' = \frac{1}{\sqrt{2n}}$$

$$\tau = \frac{1}{\sqrt{\left(2\sqrt{n}\right)}}$$

  $W_i(j)$, $W_i'(j)$, $\sigma_i(j)$, and $\sigma_i'(j)$ denote the *j*th component of the vectors W_i, W_i', σ_i, σ'_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) indicates that the random number is generated afresh for each value of *j*. *S* is a sign variance operator whose value is either positive or negative based on a normal probability distribution N(0,1).
- **Step3**: Calculate the fitness of individual population in the generation.
- **Step4**: Create a new generation by extracting members of the current population using a roulette wheel selection scheme.
- **Step5**: If the stopping criteria are satisfied stop, else go to Step 2.

C. Hybridization of the evolutionary algorithm and discrete gradient method (EADG)

In this method we are applying the Evolutionary algorithm for a certain number of generations to converge to a near optimal solution. Then we apply the Discrete gradient method as a local search with the starting point provided by the evolutionary algorithm as the best solution in the final generation. The algorithm is described as follows

- **Step1**- **Step 5** as above
- **Step6**: Calculate the fitness of an individual population in the current generation.
- **Step7**: Extract the best solution in the current generation. Let the best solution be W_b.
- **Step8**: Set the start point of the discrete gradient method $W^0 = W_b$ at $k = 0$.
- **Step9**: Set $s = 0$ and $W_s^k = W^k$.
- **Step10**: Calculate the descent direction at $W = W_s^k$ and $\partial = \partial^k$.
- **Step 11**: We calculate the least distance between the convex hull of the discrete gradients and the origin.

$$\left\|v_s^k\right\| = \min\left\{\|v\| : v \in \overline{D}_m\left(W_s^k\right)\right\}$$

- **Step 12**: Calculate the search direction (g_s^k).

$$g_s^k = -\left\|v_s^k\right\|^{-1} v_s^k$$

- **Step 13**: If $\left\|v_s^k\right\| \le \partial_k$ then set $W^{k+1} = W_s^k$, $k = k+1$ and go to Step 9.
- **Step 14**: Construct the following iteration $W^{k+1} = W_s^k + \sigma_s g_s^k$, where σ_s is defined as follows

$$\sigma_s = \arg\max\left\{\sigma \ge 0 : f\left(W_s^k + \sigma g_s^k\right) - f\left(W_s^k\right) \le c\sigma\left\|v_s^k\right\|\right\}$$

Step 15: Set $s = s+1$. Go to step 10.

III. EXPERIMENTAL RESULTS

Experiments were conducted using the benchmark data sets: Breast cancer (Wisconsin) and Heart Disease (Cleveland). Diabetes, and Liver data from the UCI Machine Learning repository. Table 1 shows the details of the individual data set used for training and testing for comparing all algorithms.

The following comparisons were made: our algorithm (EADG) with a hybrid EA and LS method (EALS), Resilient back propagation (RP) and DG method.

Table 1 Data set information

Data	Input details	Attribute information
Astral	Pattern length = 690 Training pattern = 600 Testing pattern = 90	# Input columns = 14 # Output column = 1
Breast cancer (Wisconsin)	Pattern length = 685 Training pattern = 600 Testing pattern = 85	# Input columns = 9 # Output column = 1
Cleveland	Pattern length = 297 Training pattern = 200 Testing pattern = 97	# Input columns = 13 # Output column = 1
Diabetes	Pattern length = 768 Training pattern = 700 Testing pattern = 68	# Input columns = 8 # Output column = 1
Liver	Pattern length = 345 Training pattern = 300 Testing pattern = 45	# Input columns = 6 # Output column = 1

The results of the experiments are given in Tables 2- 5. The results are compared with the results obtained by the individual methods. Time take for all the algorithms are given as well.

The following table (Table 2) shows the classification accuracy of the ANN as a percentage, CPU time in seconds for the EADG method

Table 2 Results for all data sets for EADG

Dataset	#HN	Classification Accuracy (%)	CPU Time (s)
Austral	3	91	68.4
Breast Cancer	3	100	57
Cleveland	3	90	36.2
Diabetes	5	83	53
Liver	3	88	94

The following table (Table 3) shows the classification accuracy as a percentage, CPU time in seconds for the EA and LS method [8].

Table 3 Results for all the data set for hybrid EA and LS method

Dataset	#HN	Classification Accuracy	CPU Time (s)
Austral	4	90	91
Breast Cancer	6	81	102
Cleveland	6	90	85
Diabetes	6	81	87
Liver	8	88	82

The following table (Table 4) shows the classification accuracy as a percentage and the CPU time in seconds for RP.

Table 4 Results for all the data set for RP

RP			
Dataset	#HN	Classification Accuracy	CPU Time (s)
Austral	2	87.8	29.3
Breast Cancer	4	98.8	28.7
Cleveland	4	78.4	29.3
Diabetes	3	78	30.1
Liver	7	75.6	78

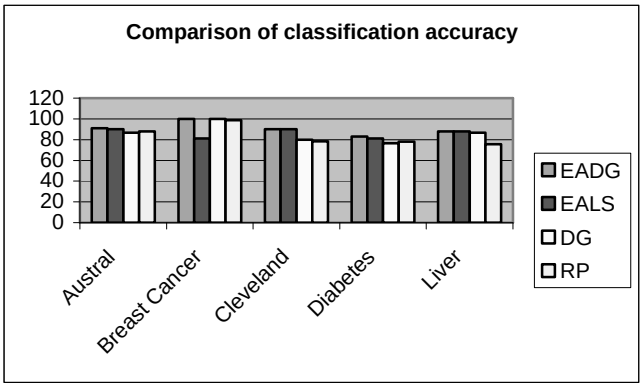
The following table (Table 5) shows the classification accuracy as a percentage and the CPU time in seconds for the DG method.

Table 5 Results for all the data set for DG

DG			
Dataset	#HN	Classification Accuracy	CPU Time (s)
Austral	2	86.7	29.69
Breast Cancer	2	100	13.12
Cleveland	2	80	13.9
Diabetes	4	76.5	59.22
Liver	3	86.7	14.07

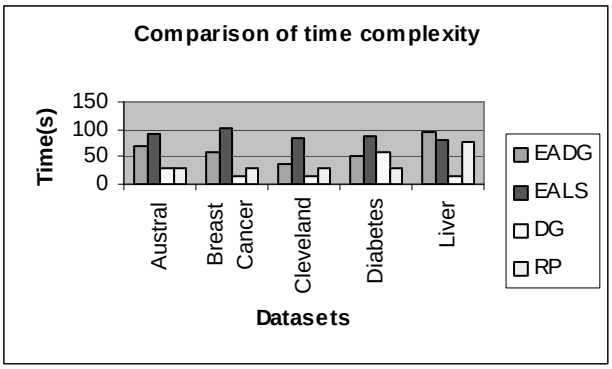
The following figure (Figure 1) shows the comparison of classification accuracy for all the four algorithms.

Figure1: Comparison of classification accuracy



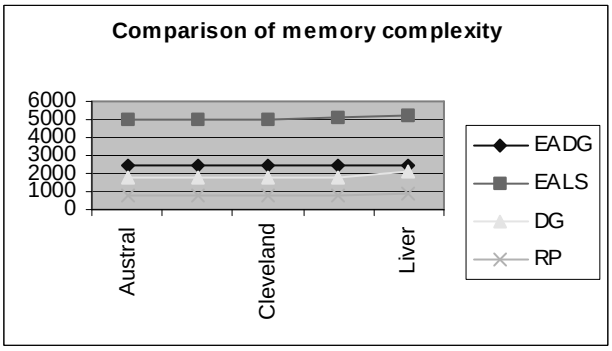
The following figure (Figure 2) shows the comparison of time complexity for all the four algorithms.

Figure2: Comparison of time complexity



The following figure (Figure 3) shows the comparison of memory complexity for all the four algorithms.

Figure3: Comparison of memory complexity



IV. CONCLUSION

The results in Tables 2-5 indicate that the hybrid EADG method achieves improvement in classification accuracy over other algorithms such as EALS, DG, and RP.

Figure 3 shows that the memory complexity is improved for the hybrid EADG method from the earlier EALS hybrid method.

The time complexity of the EADG approach for weight determination in ANNs is very favorable when compared with other algorithms.

REFERENCES

[1] O. L. Mangasarian, Mathematical programming in neural networks, *ORSA Journal on Computing*, vol. 5, pp. 349-360, 1993

[2] X. M. Zhang and Y. Q. Chen, Ray-guided global optimization method for training neural networks, *Neurocomputing*, vol. 30, pp. 333-337, 2000.

[3] T. Masters, Practical neural network recipes in C++, *Academic Press*, Boston, 1993.

[4] T. Masters, Advanced algorithms for neural networks : a C++ sourcebook, Wiley, New York, 1995.

[5]. A.M. Bagirov, Derivative-free methods for unconstrained nonsmooth optimization and its numerical analysis, *Investigacao Operacional*, vol. 19, pp. 75-93, 1999.

[6]. A.M. Bagirov, Minimization methods for one class of nonsmooth functions and calculation of semi-equilibrium prices, *Applied Optimization, vol. 30: Progress in Optimization: Contribution from Australasia.* A. Eberhard et al. (eds.), Kluwer Academic Publishers, Dordrecht, pp. 147-175, 1999.

[7]. A.M. Bagirov, A method for minimization of quasidifferentiable functions, *Optimization Methods and Software*, vol. 17,n No. 1, pp. 31-60, 2002.

[8] R. Ghosh and B. Verma, Finding Architecture and Weights for ANN Using Evolutionary Based Least Square Algorithm, *IJNS International Journal on Neural Systems*, vol. 13, no. 1, pp. 13-24, 2003.

[9] W. Schiffmann, M. Joost and R. Werner, Comparison of optimized backpropagation algorithms, *Proceedings of The European Symposium on Artificial Neural Networks*, Brussels, pp. 97-104, 1993.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representation by error propagation, *Parallel Distributed Processing, Exploring the Macro Structure of Cognition*, Cambridge, MA: MIT Press, 1986.

[11] J. H. Holland, Adaptation in natural and artificial systems, *Ann Arbor, MI*: The University of Michigan Press, 1975.

[12] I. Rechenberg, Cybernatic solution path of an experimental problem, Royal Aircraft Establishment, Library Translation no. 1122, Farnborough, Hants, U.K, Aug, 1965.

[13] D. Whitley, T. Starkweather, and C. Bogart, Genetic algorithms and neural networks - optimizing connections and connectivity, *Parallel Computing*, vol. 14, pp. 347-361, 1990.

[14] D. Montana and L. Davis, Training feed forward neural networks using genetic algorithms, *Proceedings of 11th International Joint Conference on Artificial Intelligence* IJCAI-89, vol. 1, pp. 762-767, 1989.

[15] M. Koeppen, M. Teunis, and B. Nickolay, Neural network that uses evolutionary learning, *Proceedings of IEEE International Conference on Neural Networks*, vol. 5, pp. 635-639, IEEE press, Piscstaway, NJ, USA, 1994.

[16] A. Likartsis, I. Vlachavas, and L. H. Tsoukalas, New hybrid neural genetic methodology for improving learning, *Proceedings of 9th IEEE International Conference on Tools with Artificial Intelligence*, Piscataway, NJ, USA, pp. 32-36, IEEE Press, 1997.

[17] S. Omatu and S. Deris, Stabilization of inverted pendulum by the genetic algorithm, *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation*, ETFA'96., Piscataway, NJ, USA, vol. 1, pp. 282-287, IEEE Press, 1996.

[18] S. Omatu and M. Yoshioka, Self tuning neuro PID control and applications, *Proceedings of IEEE International Conference on Systems, Man, and Cybernatics*, Picatasway, NJ, USA, vol. 3, pp. 1985-1989, IEEE Press, 1997.

[19] A. Abraham, Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, *Connectionist Models of Neurons, Learning Process, and Artificial Intelligence*, Springer-Verlag Germany, LNCS 2084, Jose Mira and Alberto Prieto (Editors), Spain, pp. 269-276, 2001.

[20] A. Abraham and B. Nath, Is Evolutionary design the solution for optimising neural networks?, *Proceedings of 5th International Conference on Cognitive and Neural Systems*

(ICCNS 2001), Published by Boston University Press, Boston, USA, 2001.
[21] G. Beliakov and A. Abraham, Global optimization of neural networks using deterministic hybrid approach, *Hybrid Information Systems, Proceedings of 1st International Workshop on Hybrid Intelligent Systems, HIS 2001*, Springer Verlag, Germany, pp 79-92, 2002.
[22] A. Bagirov, J. Yearwood, and R. Ghosh, Weight Optimization of Feedforward MLPs Using the Discrete Gradient Method, *Accepted in International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA'2004* , 12 - 14 July 2004, Gold Coast , Australia