Journal of Computer Science 1 (3): 387-394, 2005
ISSN 1549-3636

Hybridization of Neural Learning Algorithms Using Evolutionary and Discrete Gradient Approaches

Ranadhir Ghosh, John Yearwood, Moumita Ghosh and Adil Bagirov
School of Information Technology and Mathematical Sciences, University of Ballarat
P.O. Box 663, Victoria, Australia

Abstract: In this study we investigated a hybrid model based on the Discrete Gradient method and an evolutionary strategy for determining the weights in a feed forward artificial neural network. Also we discuss different variants for hybrid models using 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 best suited for global optimisation problems. Nevertheless they are cursed with longer training times and often unsuitable for real world application. 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 study we propose different fusion strategies for hybrid models combining the evolutionary strategy with the discrete gradient method to obtain an optimal solution much quicker. Three different fusion strategies are discussed: a linear hybrid model, an iterative hybrid model and a restricted local search hybrid model. Comparative results on a range of standard datasets are provided for different fusion hybrid models.

Key words: Evolutionary Algorithm, Discrete Gradient, Neural Network

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 [1-15]. 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 the length and uncertainty of the 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.* [16] 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 [17]. 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 neighbourhood 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 [18]. There were a number of researchers who used GA and EBP hybrids and reported an improvement of the algorithm over traditional GA or EBP [19, 20]. Some recent work also suggested an improvement for hybrid algorithms by running several parallel combinations of global and local search [19, 20].

Earlier work by Ghosh *et al.* [19], suggested an alternative learning methodology, which uses a hybrid technique by using evolutionary learning for the hidden layer weights and a least squares 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 the 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 thus likely 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 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 Demyanov-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 study 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.

The question that remains to be answered is: Can there be a good fusion strategy that combines these two searching modules and provides a satisfactory result? The combination of the Evolutionary algorithm with the Discrete Gradient method provides many possibilities for fusing the two different methods. The fusion strategy not only affects the classification rate but also affects the time taken to converge to the optimal or near optimal solution. It also affects the amount of memory needed to solve the problem. Hence fusion strategy plays an important role in solving these problems. In this study we propose three different fusion strategies for the proposed hybrid model combining the evolutionary strategy with the Discrete Gradient method to obtain an optimal solution more efficiently. The strategies discussed are a linear hybrid model, an iterative hybrid model and a restricted local search hybrid model. These are described in the following section. Comparative results on a range of standard datasets are provided for the different hybrid models.

MATERIALS AND METHODS

Here we describe the discrete gradient method and the evolutionary algorithm. Then we describe the hybrid method of discrete gradient and the evolutionary algorithm. Later we describe the different fusion strategies we are proposing in this study.

Discrete Gradient Method: In this section we will give a brief description of the discrete gradient method. The full description of this method can be found in [18]. The discrete gradient method can be considered as a version of the bundle method [20] when sub-gradients are replaced by their approximations-discrete gradients.

Let f be a locally Lipschitz continuous function defined on R^n. A function f is locally Lipschitz continuous on R^n if in any open bounded subset $S \subset R^n$ there exists a constant $L > 0$ such that

$$\frac{f(x)-f(y)}{\|x-y\|} \le L, \quad \forall x, y \in S. \tag{1}$$

The locally Lipschitz function f is differentiable almost everywhere and one can define for it a set of generalized gradients or a Clarke sub-differential [23], by

$$\partial f(x) = co\left\{ v \in R^{n:} : \exists \left(x^k \in D(f), \; x^k \to x, k \to \infty \right) : v = \lim_{k \to +\infty} \nabla f(x^k) \right\}, \tag{2}$$

Here $D(f)$ denotes the set where f is differentiable, co denotes the convex hull of a set and $\nabla f(x)$ stands for a gradient of the function f at a point $x \subset R^n$.
Let

$$S_1 = \{ g \in R_n : \|g\| = 1 \} \tag{3}$$

$$G = \{ e \in R^n : e = (e_1, e_2, ..., e_n), |e_j| = 1, j = 1, ..., n \} \tag{4}$$

$$P = \left\{ z(\lambda) : z(\lambda) \in R^1, \; z(\lambda) > 0, \; \lambda > 0, \lambda^{-1} z(\lambda) \to 0, \; \lambda \to 0 \right\} \tag{5}$$

$$I(g, \alpha) = \{ i \in \{1, ..., n\} : |g_i| \ge \alpha \} \tag{6}$$

where $\alpha \in \left(0, n^{-\frac{1}{2}}\right)$ is a fixed number. Here S_1 is the unit sphere, G is a set of vertices of the unit cube in R^n and P is a set of univariate positive infinitesimal functions.
We define operators $H_i^j : R^n \to R^n$ for i =1,….,n, j = 0,…,,n by the formula

$$H_i^j g = \left\{ \begin{array}{ll} (g_1, ..., g_j, 0, ..., 0) & \text{if } j < i, \\ (g_1, ..., g_{i-1}, 0, g_{i+1}, ..., g_j, 0, ..., 0) & \text{if } j \ge i \end{array} \right\} \tag{7}$$

We can see that $H_i^0 g = 0 \quad \in R^n$ for all $i = 0, ..., n$. Let $e(\beta) = (\beta e_1, \beta^2 e_2, ...\beta^n e_n), \; \beta \in (0,1]$. For $x \in R^n$ we will consider vectors

$$x_i^j(g) \equiv x_i^j(g, e, z, \lambda, \beta) = x + \lambda g - z(\lambda) H_i^j e(\beta) \tag{8}$$

where $g \in S_1$, $e \in G$, $i \in I(g, \alpha)$, $z \in P$, $\lambda > 0$, $\beta \in (0,1]$, $j = 0, ..., n, \; j \ne i$

Definition 1: The discrete gradient of the function f at the point $x \in R^n$ is the vector

$$\Gamma^i(g, e, z, \lambda, \beta) = \left(\Gamma_1^i, \Gamma_2^i, ..., \Gamma_n^i \right) \in \Re^n, \quad g \in S_1, \; i \in I(g, \alpha) \tag{9}$$

with the following coordinates:

$$\Gamma_j^i = \begin{bmatrix} z(\lambda) \\ e_j(\beta) \end{bmatrix}^{-1} \begin{bmatrix} f\left(x_i^{j-1}(g)\right) \\ -f\left(x_i^j(g)\right) \end{bmatrix}, \; j = 1, ..., n, \; j \ne i \quad (2) \tag{10}$$

$$\Gamma_i^i = (\lambda g_i)^{-1} \left[f\left(x_i^n(g)\right) - f(x) - \sum_{j=1, j \ne i}^{n} \Gamma_j^i \left(\lambda g_j - z(\lambda) e_j(\beta) \right) \right] \tag{11}$$

From the definition of the discrete gradient we can see that it is defined with respect to a given direction $g \in S_1$ and in order to calculate the discrete gradient we use step $\lambda > 0$ along this direction. The n - 1 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 that to approximate a sub-gradient of the function f. Thus the discrete gradient contains some information about the behaviour of the function f in some region around the point x.
Now we will consider the following unconstrained minimization problem:

$$\text{minimize } f(x) \text{ subject to } x \in R^n \tag{12}$$

where the function f is assumed to be locally Lipschitz continuous. We consider the discrete gradient method for solving this problem. An important step in this method is the computation of a descent direction of the objective function f. So first, we describe an algorithm for the computation of the descent direction of the function f.
Let $z \in P$, $\lambda > 0$, $\beta \in (0,1]$, the number $c \in (0,1)$ and a small enough number $\delta > 0$ be given.

Algorithm 1: An algorithm for the computation of the descent direction.

Step 1: Choose any $g^1 \in S_1$, $e \in G$, $i \in I(g^1, \alpha)$ and compute a discrete gradient $v^1 = \Gamma^i(x, g^1, e, z, \lambda, \beta)$. Set $\bar{D}_1(x) = \{v^1\}$ and $k = 1$.

Step 2: Calculate the vector $\|w^k\| = \min\{\|w\| : w \in \bar{D}_k(x)\}$. If $\|w^k\| \le \delta$ then stop. Otherwise go to Step 3.

Step 3: Calculate the search direction by $g^{k+1} = -\|w^k\|^{-1} w^k$.

Step 4: If $f(x + \lambda g^{k+1}) - f(x) \le -c\lambda\|w^k\|$, then stop. Otherwise go to Step 5.

Step 5: Calculate a discrete gradient $v^{k+1} = \Gamma^i(x, g^{k+1}, e, z, \lambda, \beta)$, $i \in I(g^{k+1}, \alpha)$, construct the set $\bar{D}_{k+1}(x) = co\{\bar{D}_k(x) \cup \{v^{k+1}\}\}$, set $k = k + 1$ and go to Step 2.

The algorithm contains steps, which deserve some explanations. In Step 1 we take any direction $g^1 \in S_1$ and calculate the first discrete gradient. In Step 2 we calculate 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 Wolfe's terminating algorithm. If this distance is less than some tolerance $\delta > 0$, the algorithm stops and we can consider this point as an approximated stationary point. Otherwise, in Step 3, a search direction is calculated. If this direction is a descent direction, the algorithm terminates, otherwise, in Step 5, we calculate a new discrete gradient with respect to this direction to improve the approximation of the set of generalized gradients. Since the discrete gradient contains some information about the behaviour of the function f in some regions around the point x this algorithm allows to find descent directions in stationary points which are not local minima (descent directions in such stationary point always exist). This property makes the discrete gradient method attractive for design of hybrid methods in global optimization. It is proved that Algorithm 1 is a terminating.

The main advantage of the Discrete gradient method is that the method approximates the descent direction with respect to the discrete gradient in a particular direction. Hence we do not require the gradient information. So the algorithm is suitable for non-smooth optimization where the derivative does not exists at certain points. The other advantage is that as Discrete gradient method is a local search technique hence it takes much less time compare to other global optimization methods. With a good initial guess the method converges to a near optimal solution with high accuracy.

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.

Fusion Strategies: There are three different fusion strategies, we are investigating in this research. All the models are described in the following subsections.

Linear Hybrid Model (LHM): The Linear Hybrid Model is so named as the two methods EA and DG are coupled linearly. In fact this corresponds to the algorithm described earlier.

In this model we are applying the Evolutionary algorithm for a certain number of generation to converge to a near optimal solution. Then we applying the Discrete gradient method as local search with the starting point provided by the evolutionary algorithm as the best solution in the final generation. The discrete gradient method generates the final optimal output. A sample genotype used for the LHM for *n* input, *h* hidden units, *k* output and *pat* number of patterns can be written as:

$$\begin{vmatrix} w^h_{11}\mu^h_{11} \ldots w^h_{1n}\mu^h_{1n}\, w^h_{21}\mu^h_{21} \ldots w^h_{2n}\mu^h_{2n} \ldots w^h_{h1}\mu^h_{h1} \ldots w^h_{hn}\mu^h_{hn} \\ w^o_{11}\mu^o_{11} \ldots w^o_{1h}\mu^o_{1h}\, w^o_{21}\mu^o_{21} \ldots w^o_{2h}\mu^o_{2h} \ldots w^o_{k1}\mu^o_{k1} \ldots w^o_{kh}\mu^o_{kh} \end{vmatrix}$$

where, range(w) initially is set in the closed interval [-1+1]. The evolutionary algorithm starts by initialising the population pool and tries to find the near optimal solution over few generations. The fitness of the population is determined as the classification error on the validation dataset. After getting a near optimal solution the evolutionary algorithm passes the solution to the Discrete Gradient method. The discrete gradient method takes the near optimal solution as starting point and tries to generate the optimal set of weights for the neural network.

A good staring point provided by the Evolutionary algorithm helps the discrete gradient method to improve the quality of the solution. Also the evolutionary algorithm is running for only a few generation, hence the time taken by that algorithm to generate the staring point for the Discrete gradient method is less than the normal evolutionary algorithm would take for the

whole process. Hence the time taken by the hybrid model to converge is much less that of the Evolutionary algorithm. The disadvantage of the model is that it depends on the convergence quality of the Evolutionary Algorithm. As the Evolutionary algorithm is running for a few generations, sometimes it cannot converge to a good near optimal solution. Hence it often fails to serve as a good staring point to the Discrete gradient method. As the convergence of discrete gradient method depends on the staring point, hence the quality of the solution is not always good in the Linear Model.

Iterative Hybrid Model (IHM): The Iterative Hybrid model is so named because the two methods are coupled in each iteration.

In this model we are applying the Discrete gradient method to all the individuals in the population in each generation. This algorithm is equivalent to multiple discrete gradient solutions. Each solution is involved in a competitive selection after each selection and the Evolutionary algorithm rejects the poor solution. The solutions with high fitness are selected and mutated to serve as the staring point of the Discrete gradient methods in the next generation. The stopping criteria of this model are the number of generation and the classification accuracy of the best solution in a generation. A sample genotype used for the IHM for *n* input, *h* hidden units, *k* output and *pat* number of patterns can be written as

$$\left|\begin{matrix} w^h_{11}\mu^h_{11}...w^h_{1n}\mu^h_{1n}w^h_{21}\mu^h_{21}...w^h_{2n}\mu^h_{2n}...w^h_{h1}\mu^h_{h1}...w^h_{hn}\mu^h_{hn} \\ w^o_{11}\mu^o_{11}...w^o_{1h}\mu^o_{1h}w^o_{21}\mu^o_{21}...w^o_{2h}\mu^o_{2h}...w^o_{k1}\mu^o_{k1}...w^o_{kh}\mu^o_{kh} \end{matrix}\right|$$

where, range(w) initially is set in the closed interval [-1+1]. The Iterative Hybrid algorithm starts by initialising the population pool. Then it applies the Discrete gradient method to generate the optimal solutions by the method with those staring points. Then it applies evolutionary strategies to select the better solutions and generates the next generation.

The iterative hybrid algorithm generates multiple optimal solutions and selects the best one. It decreases the chance for the discrete gradient method to get stuck in a local minimum. Even if it gets stuck in certain cases the evolutionary algorithm is capable of correcting the point either through recombination or by rejecting the point or just passing it to a starting point and generates better staring points in next generation. Hence the quality of the solution in this model should be much better than the Linear Hybrid model in most of the cases. Also the chances of getting an optimal solution is much higher than the Evolutionary algorithm, hence the time taken by this hybrid algorithm is less than the normal evolutionary algorithm takes for the whole process. The disadvantage of the model is that it takes more time to converge than the linear hybrid method. As the discrete gradient method is run for each population in each generation, the Iterative Hybrid Method is much slower than the previous model.

Restricted Local Search Hybrid Model (RLSHM): The weight variables for each layer are found using a hybrid method, which uses the evolutionary algorithm and the Discrete gradient method. The Restricted Local Search Hybrid model is an extension of the iterative hybrid model when the local search method involved in the hybridisation is limited to search in one layer of the neural network. The architecture is shown in Fig. 1. The Evolutionary algorithm is applied to the hidden layer weight and the discrete gradient method is 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].

The Evolutionary algorithm is involved in modifying the weights for the hidden layer. The Discrete gradient method is involved in modifying the weights for the output layer. The hybrid algorithm first initialises all the weights using a uniform distribution on a closed interval range of [-1, +1]. The evolutionary algorithm uses only the hidden layer weights. A sample genotype used for the RLSHM for *n* input, *h* hidden units, *k* output and *pat* number of patterns can be written as

$$\left|w^h_{11}\mu^h_{11}...w^h_{1n}\mu^h_{1n}w^h_{21}\mu^h_{21}...w^h_{2n}\mu^h_{2n}...w^h_{h1}\mu^h_{h1}...w^h_{hn}\mu^h_{hn}\right|$$

The discrete gradient is then applied in the output layer with the hidden layer weights fixed. The fitness is then calculated using the weight represented by the population genome and the Discrete gradient output of the output layer weights. The solution with high fitness are selected and in the next generation. The stopping criteria of this model are the number of generation and the classification accuracy of the best solution in a generation. One of the most important attributes in this hybridisation is the application of the DG method for the output layer only. Hence the fitness of the chromosome can be affected by breaking into two halves.

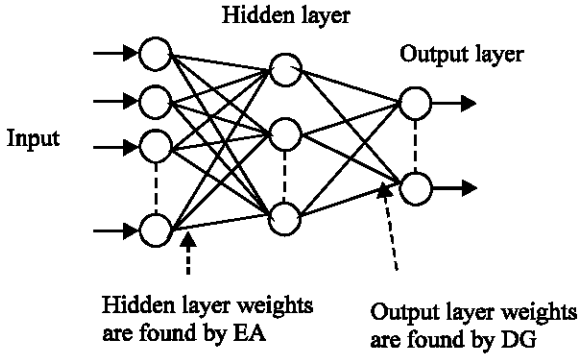

Fig. 1: ANN Architecture for RLSHM Model

Table 1: Data Set Information

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

Table 2: Classification Accuracy Results for All Data Sets for Fusions of EA and DG

Dataset	Test Classification Accuracy (%)					
	LHM	IHM	RLSHM	EA	RP	DG
Austral	91	92.2	90	90	87.8	86.7
Breast Cancer	100	100	99	85.5	98.8	100
Cleveland	90	90.7	90.7	89.7	78.4	80
Diabetes	83.8	85.2	82.3	81	78	76.5
Liver	88.8	93.3	93.3	86.7	75.6	86.7

Table 3: Time Complexity Results for All Data Sets for Fusions of EA and DG

Dataset	CPU Time (s)					
	LHM	IHM	RLSHM	EA	RP	DG
Austral	68.4	115	101	94	29.3	29.69
Breast cancer	57	96.4	87.4	68	28.7	13.12
Cleveland	36.2	58.4	44.2	44	29.3	13.9
Diabetes	53	78.1	69	68.4	30.1	59.22
Liver	94	139	73	101	78	14.07

EXPERIMENTAL RESULTS

Dataset: 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.

Results: Table 2 and 3 show the classification accuracy as a percentage and the time complexity of the ANN for all methods and data sets.

Figure 2 and 3 show a comparison of classification accuracy and the time complexity for all the algorithms. Figure 4 shows the comparison of memory complexity for all the algorithms.

Analysis of Convergence: The convergence property graph (Fig. 5) shows that steady convergence is obtained using the Linear Hybrid Model. However the convergence time is much greater than the convergence times of IHM and RLSHM. The convergence property for the iterative hybrid model has some sudden decrease by applying the DG method and do not have the ups and downs such as seen in the RLSHM. It can be seen that the application of the DG method guarantees a quantum improvement for the solution point.

Analysis of Fitness Breaking for RLSHM: Figure 6 shows the fitness distribution for the 10 best populations before the DG method is applied. Figure 7 shows the fitness distribution for the same population after applying the DG method. After every generation of the evolutionary algorithm, the fitness for the population pool is calculated by the evolutionary algorithm. Then the fitness of the chromosome is calculated by breaking it into two halves and taking the first half and then combining it with the output layer weights (by calling the least square function).

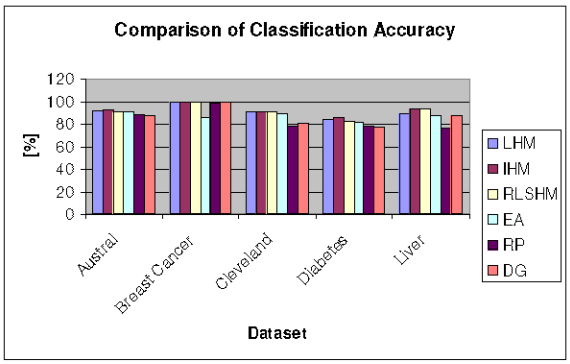

Fig. 2: Comparison of Classification

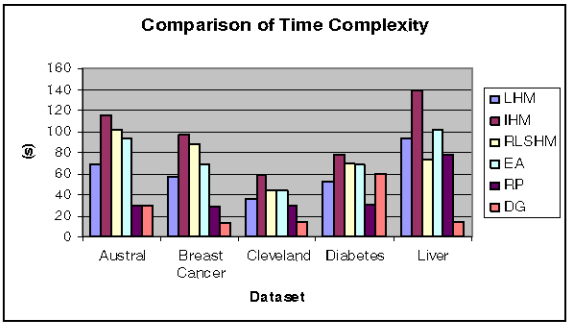

Fig. 3: Comparison of Time Complexity

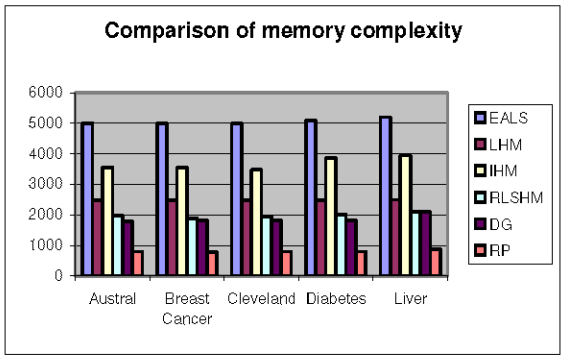

Fig. 4: Comparison of Memory Complexities for All Algorithms

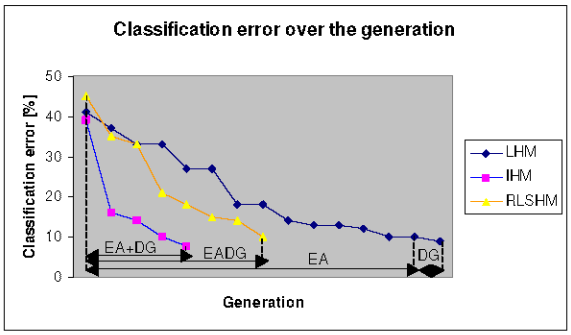

Fig. 5: Comparison of Convergence Property for All Algorithms

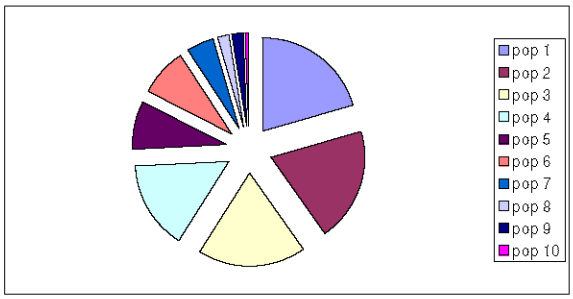

Fig. 6: Fitness Distribution of EA before Applying DG

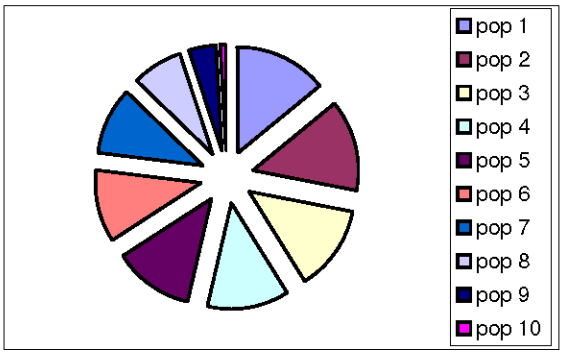

Fig. 7: Fitness Distribution of EA After Applying DG

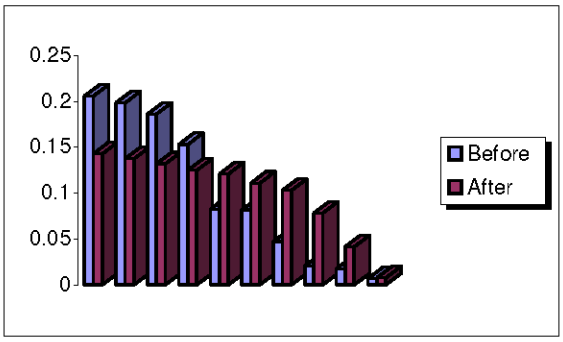

Fig. 8: Ranking of the Population Based on Fitness

The comparisons of the fitness values are reported. (Population number is given based on the rank, rather than the raw number of the population obtained from the program). Figure 6 and 7 show that the fitness of the gene (particularly for the chromosomes with very high fitness) is not affected much when it is broken and combined with the DG method.

Further Fig. 8 shows that the ranking based on their fitness value remains unaffected. So, even when the upper half characteristic is replaced by the DG weights, and only the lower half characteristic is considered, it does not distort the fitness and affect their rank. The fitness based on the lower half characteristic of the chromosome has almost the same rank in the population pool, with the fitness based on the full-length chromosome.

CONCLUSION

In this work we have investigated a hybrid model based on the Discrete Gradient method and an evolutionary strategy for determining the weights in a feed forward artificial neural network. We have used three different fusion strategies for hybridising EA and DG and compared their performance in terms of classification accuracy and time complexity. The results indicate that there is generally an improvement in classification accuracy, time complexity and memory complexity for hybridisation. Time complexity results are more complex but indicate that some fusion strategies can also improve performance over single approaches whilst others may not or indeed may decrease performance. In the RLSHM fusion, it is interesting to find that the decomposition of the chromosome does not distort the fitness rank of the generations. In future research, we will explore a slight variant of the iterative hybrid model, where the DG method can be applied for the whole population, thus exploring a bigger range of initial solution points.

REFERENCES

1. Mangasarian, O.L., 1993. Mathematical programming in neural networks. ORSA J. Computing, 5: 349-360.
2. Zhang, X.M. and Y.Q. Chen, 2000. Ray-guided global optimization method for training neural networks. Neurocomputing, 30: 333-337.
3. Masters, T., 1993. Practical Neural Network Recipes in C++. Academic Press, Boston.
4. Masters, T., 1995. Advanced Algorithms for Neural Networks: A C++ Sourcebook. Wiley, New York.
5. Whitley, D., T. Starkweather and C. Bogart, 1990. Genetic algorithms and neural networks-optimizing connections and connectivity. Parallel Computing, 14: 347-361.
6. Montana, D. and L. Davis, 1989. Training feed forward neural networks using genetic algorithms. Proceedings of 11th Intl. Joint Conference on Artificial Intelligence IJCAI-89, 1: 762-767.
7. Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.
8. Belew, R.K., J. McInerney and N.N. Schraudolph, 1991. Evolving networks: using genetic algorith, with connectionist learning. Technical Report #CS90-174 (Revised), Computer Science and Engineering Department (C-014), University of California at San Diego, La Jolla, CA 92093, USA.
9. Topchy, A.P. and O.A. Lebedko, 1997. Neural network training by means of cooperative evolutionary search. Nuclear Instruments and Methods in Physics Research, Section A: Accelerators. Spectometers, Detectors and Associated Equipment, 389: 240-241.
10. Koeppen, M., M. Teunis and B. Nickolay, 1994. Neural Network that Uses Evolutionary Learning. Proceedings of IEEE Intl. Conference on Neural Networks, IEEE Press, Piscstaway, NJ, USA, 5: 635-639.
11. Likartsis, A., I. Vlachavas and L.H. Tsoukalas, 1997. New hybrid neural genetic methodology for improving learning. Proceedings of 9th IEEE Intl. Conference on Tools with Artificial Intelligence, Piscataway, NJ, USA, pp: 32-36.
12. Omatu, S. and S. Deris, 1996. Stabilization of inverted pendulum by the genetic algorithm. Proceedings of IEEE Conference on Emerging Technologies and Factory Automation, ETFA'96., Piscataway, NJ, USA, IEEE Press, 1: 282-287.
13. Omatu, S. and M. Yoshioka, 1997. Self tuning neuro PID control and applications. Proceedings of IEEE Intl. Conference on Systems, Man and Cybernatics, Picatasway, NJ, USA, IEEE Press, 3: 1985-1989.
14. Abraham, A., 2001. 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 (Eds.), Spain, pp: 269-276.
15. Abraham, A. and B. Nath, 2001. Is evolutionary design the solution for optimising neural networks? Proceedings of 5th Intl. Conference on Cognitive and Neural Systems (ICCNS 2001), Published by Boston University Press, Boston, USA.
16. Rumelhart, D.E. *et al.*, 1988. Parallel Distributed Processing. Cambridge, MA, MIT Press, Vol: 1.
17. Rechenberg, I., 1965. Cybernetic solution of an experiental problem. Royal Aircraft Establishment, Library Transaction no. 1122, Farnborough, Hants, U.K.
18. Beliakov, G. and A. Abraham, 2002. Global optimization of neural networks using deterministic hybrid approach. Hybrid Information Systems, Proceedings of 1st Intl. Workshop on Hybrid Intelligent Systems, HIS 2001, Springer Verlag, Germany, pp: 79-92.
19. Ghosh, R. and B. Verma, 2003. Finding architecture and weights for ANN using evolutionary based least square algorithm. Intl. J. Neural Systems, 13: 13-24.
20. Bagirov, A.M., 1999. Derivative-free methods for unconstrained nonsmooth optimization and its numerical analysis. Investigacao Operacional, 19: 75-93.
21. Bagirov, A.M., 2002. A method for minimization of quasidifferentiable functions. Optimization Methods and Software, 17: 31-60.