Selection of Optimal Set of Weights in a Layered Network Using Genetic Algorithms

SANKAR K. PAL

and

DINABANDHU BHANDARI

Machine Intelligence Unit, Indian Statistical Institute, 203 B. T. Road, Calcutta 700 035, India

Communicated by Abe Kandel

ABSTRACT

Genetic algorithms represent a class of highly parallel robust adaptive search processes for solving a wide range of optimization and machine learning problems. The present work is an attempt to demonstrate their effectiveness to search a global optimal solution to select a decision boundary for a pattern recognition problem using a multilayer perceptron. The proposed method incorporates a new concept of nonlinear selection for creating mating pools and a weighted error as a fitness function. Since there is no need for the backpropagation technique, the algorithm is computationally efficient and avoids all the drawbacks of the backpropagation algorithm. Moreover, it does not depend on the sequence of the training data. The performance of the method along with the convergence has been experimentally demonstrated for both linearly separable and nonseparable pattern classes.

1. INTRODUCTION

One of the remarkable developments of the artificial neural network (ANN) is classifier design, viz. multilayer perceptron (MLP) [1]. Initially, in designing a classifier, a set of objects with known class levels is used for training (learning). The system is then asked to classify an unknown object based on the information acquired during training. The training method in MLP is supervised and is accomplished through the well known backpropagation technique.

It has been found that MLP can model highly complex decision boundaries for pattern classification, but has the major problem of getting stuck at local optimal solutions during training. Moreover, the method of correcting the weights through backpropagation of error is time consuming. Another drawback of the existing backpropagation learning in MLP from the point of view of classifier design is that its convergence at the proper decision boundary depends on the sequence of the input data without taking into account the global effect of the training set.

Researchers are now trying to incorporate genetic algorithms [2, 3] in designing and learning neural networks. In [4], Bornholdt and Graudeng described a model for a genetically altered neural net. Muhlenbein [5] proposed a genetically inspired modular neural network instead of MLP in the task domain of Boolean functions. In [6], Whitley et al. developed a genetic algorithmic approach for optimizing connections and connectivity of neural networks and tested it by pruning a fully connected two-bit adder and XOR. They discussed new developments of genetic algorithms and also proposed a new mutation process, called adaptive mutation, to maintain the genetic diversity in the population.

In this article an attempt has been made to incorporate the genetic algorithms in selecting the global optimal set of weights in an MLP for pattern recognition. The proposed method is highly parallel, robust, and avoids the conventional backpropagation technique, thereby reducing the computational overhead. Since the algorithm does not need the backpropagation technique, its performance does not depend on the sequence of training data and the choice of learning rate is no longer required. Unlike the error function used in conventional learning, the proposed algorithm uses a weighted mean square error (depending on the probability of occurrence of the classes) as the basis of the evaluation function. A new concept of the nonlinear selection process is introduced for creating mating pools. A comparative study has also been made regarding the computational time required for the parallel implementation of the conventional backpropagation technique and the proposed method.

The algorithm is implemented on two different sets of linearly separable/nonseparable patterns. The performance of the method along with the convergence is provided for different training sets, population sizes, and initial populations.

2. GENETIC ALGORITHMS: BASIC PRINCIPLES AND FEATURES

Genetic algorithms (GAs) [2, 3] are highly parallel and adaptive search and machine learning processes based on the mechanics of natural selection and the natural genetic system. GAs are capable of solving a wide range of complex optimization problems [5, 7–9] using genetic operators (reproduction/selection, crossover, and mutation) on coded solutions (strings/chromosomes) in an iterative fashion. They efficiently exploit historical information to speculate on new search points with expected improved performance. GAs deal simultaneously with multiple points (called population), not a single point, which helps to find the global optimal solution without getting stuck at local optima. GAs are theoretically and empirically proven to provide robust search in complex spaces, even if the searching (e.g., optimization) function spaces are not smooth or continuous, which are very difficult (sometimes impossible) to handle using calculus-based methods. GAs are also blind, that is, they use only the payoff or penalty (i.e., objective) function and do not need any other auxiliary information. A schematic diagram of the basic structures of a genetic algorithm is shown in Figure 1.

Reproduction is a process in which individual strings are copied according to their objective function values, f, called the fitness function. This operation is an artificial version of natural selection, a Darwinian survival of the fittest among string creatures. More highly fitted strings have a higher number of offspring in the succeeding generation. These strings are then entered into a mating pool, a tentative new population, for further genetic operator action.

The crossover operation generates offspring for the new generation using the strings (parents) selected randomly from the mating pool. The crossover may be thought of as an information exchange procedure between two potential strings and it produces a pair of offspring.

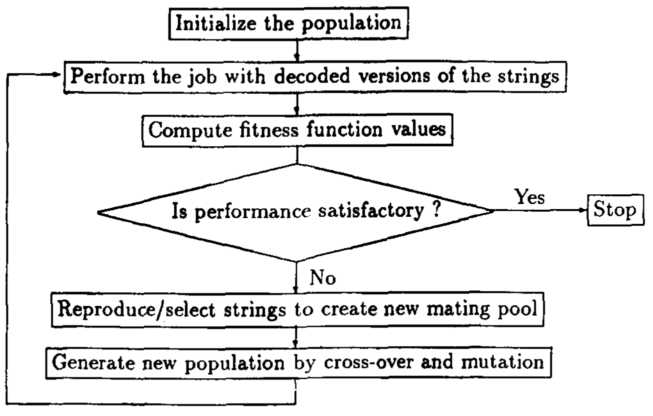

Initialize the population
Perform the job with decoded versions of the strings
Compute fitness function values
Is performance satisfactory ?
Yes
Stop
No
Reproduce/select strings to create new mating pool
Generate new population by cross-over and mutation

Fig. 1. Basic steps of the genetic algorithm.

In a simple GA, mutation is an occasional random alteration of the value of a string position. The mutation operator plays a secondary role in the simple GA. The frequency of mutation to obtain good results is on the order of one per thousand bits (position) [3]. Like an insurance policy, it helps to prevent the irrecoverable loss of potentially important genetic material.

Let us now describe, in brief, multilayer perceptron (MLP) before we demonstrate the capability of GAs to select its optimal set of connection strengths (weights) for classifying patterns.

3. MULTILAYER PERCEPTRON AND CLASSIFIER DESIGN

One of the most exciting developments during the early days of pattern recognition was the *perceptron*. It may be defined as a network of elementary processors arranged in a manner reminiscent of biological neural nets that are able to learn how to recognize and classify patterns in an autonomous manner. In such a system, the processors are simple linear elements arranged in one layer. This classical (single layer) perceptron, given two classes of patterns, attempts to find a linear decision boundary separating the two classes. If the two sets of patterns are linearly separable, the perceptron algorithm is guaranteed to find a separating hyperplane in a finite number of steps. However, if the pattern space is not linearly separable, the perceptron fails and it is not known when to terminate the algorithm in order to get a proper decision boundary. Thus, a single layer perceptron is inadequate for situations with multiple classes and nonlinear separating boundaries. This motivated the invention of a multilayer network (MLN) with nonlinear learning algorithms that is known as the *multilayer perceptron* (MLP) [1]. The MLNs can produce boundaries for complex linearly nonseparable classes.

A schematic representation of a multilayer perceptron (MLP) is given in Figure 2. The outputs of nodes in one layer are transmitted to nodes in another layer via links/connections that amplify, attenuate, or inhibit such outputs through weighting factors. The total input to the ith unit (node) of any layer, except the input layer, is

$$U_i = \sum_j W_{ij} V_j. \tag{1}$$

Here V_j is the output of the jth unit of the previous layer and W_{ij} is the connection weight between the ith node of one layer and the jth node of

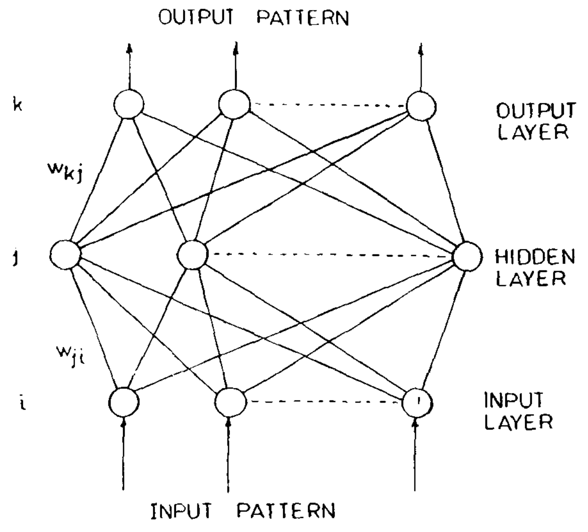


Fig. 2. Schematic representation of an MLP.

the previous layer. The output of a node i is

$$V_i = g(U_i), \tag{2}$$

where $g(\cdot)$ is the activation function. The activation function is mostly sigmoidal, with the form

$$V_j = \frac{1}{1 + \exp(-(U_j - \theta_j))}, \tag{3}$$

where θ_j is the threshold associated with the node.

The learning (training) system in the multilayer perceptron (MLP) is supervised through the well known backpropagation algorithm, which is based on the gradient descent technique. In this method, the correct set of weights is obtained by reducing the error. This is achieved by moving in the direction of the negative gradient of the error function (E), defined in (4).

During training, each pattern of a training set T is used in succession to clamp the input and output layers of the network. Feature values of the input patterns are clamped to the input nodes, whereas the output nodes are clamped to class labels. The network gets (sequentially) patterns like

$\mathbf{x}_k = \{x_{kl}\} \in T$, where x_{kl} is the lth component of the vector $\mathbf{x}_k$, as input. The input is then passed on to the output layer via the hidden layers to produce output $\mathbf{V}_k = \{V_{kj}\}$. In general, the outputs $\{V_{kj}\}$ will not be the same as the targeted or desired values $\{t_{kj}\}$. For a pattern $\mathbf{x}_k$, the error (E_k) is calculated as

$$E_k = \sum_j (t_{kj} - V_{kj})^2. \tag{4}$$

The overall error (E) for the training set T is calculated as

$$E = \frac{1}{K} \sum_l E_k = \frac{1}{K} \sum_k \sum_j (t_{kj} - V_{kj})^2, \tag{5}$$

where K $(=|T|)$ is the number of training samples.

For every input pattern $\mathbf{x}_k$, the corresponding error E_k [(4)] is backpropagated to modify the weights so that the desired outputs $\{t_{kj}\}$ are obtained at the output nodes. A sequence of forward and backward passes is made for each input pattern of the training set for the stabilization of the network. After the network has been settled down, the weights specify the boundaries of the classes. Thus, learning in an MLP for classifier design involves finding an optimum set of weights (minimizing the error) that take information from the given training data set in order to classify unknown patterns properly.

Two problems with the existing learning process of the MLP are (1) it is computationally expensive and (2) the backpropagation process does not guarantee the convergence at the global optimum, i.e., one may get stuck to local optima of the error function. It is also to be mentioned here that the backpropagation is a gradient descent method and, therefore, it needs the objective functions to be derivable. In various problems of pattern recognition, particularly in image processing, the objective functions are not so.

In the conventional learning strategy for classifier design, the weights are usually updated for each pattern instead of accumulating information about the whole training set. Therefore, the convergence of the network depends on the sequence of the input pattern (training set). Besides these, there is a difficulty in selecting the learning rate (ϵ) [1]. A high value of ϵ may make the network oscillatory, and a low value results in slow convergence.

In the following section, we propose a methodology, based on genetic algorithms, which is capable of searching the global optimum solution. The proposed algorithm selects the appropriate weights randomly from a set of potential solutions and there is no need of the backpropagation technique. As a result, the algorithm is computationally less expensive, and the selection of learning rate ϵ and the derivability of objective function are no longer required. Moreover, it considers only the global effect of training set in updating weights; therefore, it does not depend on the sequence of the training data set.

4. INCORPORATION OF GENETIC ALGORITHMS IN MLP

Let us consider an MLP having L layers, including the input (1st) and output (Lth) layers. Let p_i be the number of nodes in the ith $(1 \leqslant i \leqslant L)$ layer. Let a threshold (bias) be associated with each node (except the nodes in the input layer). Therefore, the number of parameters in such a network is

$$P = \sum_{i=1}^{L-1} p_{i+1}(p_i + 1). \tag{6}$$

For example, for a two-layered network ($L = 2$) having two nodes in each layer ($p_1 = p_2 = 2$), the number of parameters is $P = 2 \times (2+1) = 6$.

In the following subsections we shall describe the basic components of genetic algorithms in the context of assigning link weights during learning/training in an MLP.

4.1. CHROMOSOMAL REPRESENTATION OF WEIGHTS AND INITIAL POPULATION

GAs search for the global, near optimal solution under the complete lack of knowledge about the search spaces. Usually in GAs, the initial approximations are random binary strings. A binary string of length Pq can be considered as a chromosomal representation of the parameter set. Here, the first q bits are assumed to be the representative of the first parameter, the next q bits are for the second parameter, and so on. Therefore, in the previous example, when $q = 10$ the representation of the parameter set is

$$\begin{array}{cccc} 1100010101 & 0100011010 & \cdots & 0111110001 \\ \text{par}_1 & \text{par}_2 & \cdots & \text{par}_6 \end{array}$$

Each substring of length q is then decoded into $[-1,1]$ and multiplied by some suitable constant to make the parameter values lie in some desired domains.

It has already been mentioned earlier that GAs start with multiple points (approximations), not with a single point, unlike other search processes, and generate an improved set of approximations from the potential strings of the previous generation. The set of strings in a generation is called a population. A set of random binary strings each of length Pq (q bits for each parameter) can be considered as an initial population.

Selection of the size (N) of the population, i.e., the number of strings in a population is an important task in GAs. The size may be fixed in each generation or varied with generation. One can keep it constant by ignoring the strings that have lower fitness values and taking into consideration the offspring produced from the potential (highly fitted) strings of the previous population. It is to be noted that the greater is the number of strings in the population, the higher is the processing time required for each iteration and the smaller is the number of generations (iterations) to be executed for convergence.

4.2. *FITNESS FUNCTION*

In GA the objective/fitness function is the final arbiter of the string creators. Here, a highly fitted string should have higher fitness value and it should result in low classification error. Therefore, any decreasing function $F(E)$ of overall error E [(5)] can be considered as the fitness function. For example, $F(E)$ can be taken as

$$F(E) = E_{\max} - E, \tag{7}$$

where $E_{\max}$ is the maximum possible value of the error function.

It is also to be mentioned here that the number of patterns of each class is usually different. Therefore, during training, the contributions of different classes in evaluating overall error are different and, as a result, may produce improper decision boundaries. In order to avoid this situation, we introduce a weighted sum [10] in (5) such as

$$E_W = \frac{1}{K} \sum_k \sum_j (t_{kj} - V_{kj})^2 \times \alpha_j, \tag{8}$$

with $\alpha_j = 1 - p_j = 1 - |c_j|/K$. p_j is the apriori probability of class C_j, $|c_j|$ is the number of training patterns of the jth class, and K is the size of the training data set. The weighting coefficient α_j takes care of the effect of unequal proportion of training samples in positioning the decision boundary.

4.3. GENETIC OPERATORS

4.3.1. Reproduction/Selection

The reproduction (selection) process is executed (as described in Section 2) by copying the individual strings, according to their fitness function values, into the mating pool for the purpose of crossover and mutation operations. Let f_i be the fitness value obtained for a training set T corresponding to the ith string S_i, $i = 1,2,\ldots,N$. Then the mating pool will consist of n_i copies of S_i, where,

$$n_i = \frac{f_i}{\Sigma f_i} N. \tag{9}$$

Note that n_i may not be an integer. In that case, we round it off such that $\Sigma_i n_i = N$.

The purpose of the above selection procedure is to mimic natural selection: Darwinian survival of the fittest. In other words, the procedure allows generation of more offspring for the next generation from potential strings (strings with high fitness values).

Nonlinear selection. The above linear selection process improves the performance by maintaining the potential of the population, but it is not enough to improve the search speed and to maintain the genetic homogeneity in the population. Moreover, the natural selection process is not linear. In order to take these factors into account, one may introduce some nonlinearity in generating the mating pool such that it would increase the genetic homogeneity in the pool by reducing the discrepancy between strings with higher (or lower) fitness values. At the same time, we also need to increase the number of copies of a string having high fitness value in the mating pool and to decrease that of a string having low fitness value. This can be implemented using a nonlinear function, e.g., S-type function [11] of the form

$$S(x: f_{\min}, f_{\max}) = \begin{cases} 0 & \text{if } x \leqslant f_{\min} \\ 2\dfrac{(x - f_{\min})^2}{(f_{\max} - f_{\min})^2} & \text{if } f_{\min} \leqslant x \leqslant (f_{\min} + f_{\max})/2 \\ 1 - 2\dfrac{(f_{\max} - x)^2}{(f_{\max} - f_{\min})^2} & \text{if } (f_{\min} + f_{\max})/2 \leqslant x \leqslant f_{\max} \\ 1 & \text{otherwise.} \end{cases} \qquad (10)$$

Here, $f_{\min}$ and $f_{\max}$ are, respectively, the minimum and maximum fitness function values corresponding to the strings of the considered population.

This can be viewed as putting a soft threshold over the strings to employ their unequal importance in generating a mating pool. This nonlinear selection method therefore enables maintainence of genetic homogeneity besides producing a potential mating pool, thereby enhancing the chance of crossover between two highly fitted strings for future processes.

4.3.2. Crossover

Since the size of the parameter set, and consequently the length of the chromosomes, is not small, it is intuitive that the single point crossover operation (as described in Section 2) may not be useful for fast convergence. Therefore, instead of applying a crossover operation at a single point over the entire string, we applied this operation on each substring (chromosomal representation of an individual parameter). The proposed multiple point crossover operation is demonstrated below for the substring length $q = 10$. Let

$$\begin{aligned} a &= 1100010101 \quad 0100011010 \quad \cdots \quad 0111110001, \\ b &= 1000101110 \quad 1110110001 \quad \cdots \quad 0011010100 \end{aligned}$$

be two strings (parents) selected for crossover. Let the random number generated by the crossover operation be 7,5,...,4. Then the newly produced offspring will be

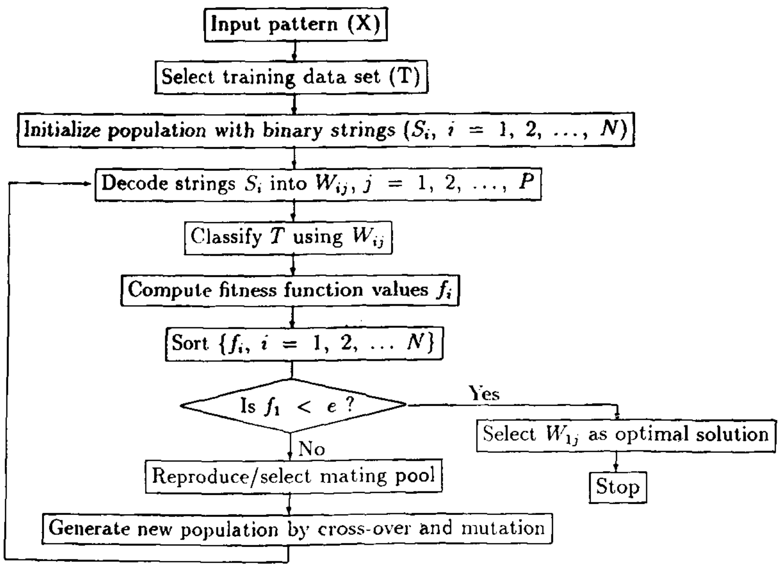
$$a' = 1100010110 \quad 0100010001 \quad \cdots \quad 0111010100,$$
$$b' = 1000101101 \quad 1110111010 \quad \cdots \quad 0011110001.$$

4.3.3. *Mutation*

The mutation operation is performed with very low probability (P_{mut}), but it is difficult to determine the probability of performing this operation in order to produce good result. We have chosen P_{mut} in the range of 0.002 to 0.001. A random bit position of a random string is selected for mutation and the status of the bit at that position is reversed (i.e., 0 is replaced by 1 and vice versa).

4.4. *ALGORITHM*

The block diagram of the proposed algorithm is presented in Figure 3. Given the pattern classes, a training data set T is selected for learning the neural network parameters. Initially, a binary string, generated randomly,



Input pattern (X)
Select training data set (T)
Initialize population with binary strings (S_i, i = 1, 2, ..., N)
Decode strings S_i into W_{ij}, j = 1, 2, ..., P
Classify T using W_{ij}
Compute fitness function values f_i
Sort {f_i, i = 1, 2, ... N}
Is f_1 < e ?
Yes
No
Select W_{1j} as optimal solution
Stop
Reproduce/select mating pool
Generate new population by cross-over and mutation

Fig. 3. Block diagram of the proposed algorithm.

of length pq (substrings of length q are taken for each parameter) is considered as the chromosomal (string) representation for the parameter set associated with the network. A set of N such strings is considered as the initial population. In our experiment, we have assumed $N = 50$. The substrings of a string are then decoded into a real number in the interval $[-1, 1]$ and multiplied by some constant to make them lie in some desired domain. The fitness value corresponding to each string is calculated [using (7)] for the entire training set. For a pattern $\mathbf{x}_k$, if the output obtained at the nodes of the output layer is V_{k1} and V_{k2}, then we say $\mathbf{x}_k$ is in class 1 if $V_{k1} \geqslant V_{k2}$; otherwise $\mathbf{x}_k$ is in class 2.

A mating pool is created using the reproduction operation (explained in Section 4.3.1) with highly fitted strings and a new population of size N for the next generation is then generated by the crossover and mutation operations on the strings selected randomly from the new mating pool.

The learning process of the network is then repeated with the parameter sets corresponding to this new population for the same training set T of patterns and, consequently, a new population is generated further. The process terminates when the minimum value of the error over a population becomes less than some small preassigned value e. The decoded version of the string having minimum error value then represents the optimum parameter set for the network.

5. COMPARISON OF COMPUTATIONAL TIME

Here we have made an attempt to determine the computational time required for parallel implementation of the conventional backpropagation technique and the proposed method. In determining the time of computation, we have assumed that the time required for *multiplication, function call, and exchange* are the same and are equal to 1 unit (say).

Let us consider an MLP that has L layers and let N_d be the size of the training data set. For the backpropagation technique, let

Δt = average time taken by each node during forward pass,

ΔT = average time taken for weight correction in each link,

N_i = the number of iterations required to get the desired output.

Then, the total time required is

$$T_{\mathrm{BP}} = (L \cdot \Delta t + (L-1) \cdot \Delta T) \times N_d \times N_i. \qquad (11)$$

Now, $\Delta t = 2$ (one multiplication step and one for function call) and $\Delta T = 3$ (two multiplication steps and one for function call and multiplications simultaneously). Hence,

$$T_{\mathrm{BP}} = (5L - 3) \times N_d \times N_i. \tag{12}$$

For the proposed algorithm, let the same training data set be used for learning and let N_p be the population size (number of parameter strings considered),

ΔT_D = time required for decoding the strings,

ΔT_C = time taken for crossover,

ΔT_M = time taken for mutation,

ΔT_S = time taken for sorting the fitness values,

N_g = the number of iterations required to get the desired output.

Then the total time required is

$$T_{\mathrm{GA}} = (L \cdot \Delta t \times N_d + \Delta T_D + \Delta T_C + \Delta T_M + \Delta T_S) \times N_g. \tag{13}$$

Again, $\Delta T_D = 1$ (one multiplication step), $\Delta T_C = 1$ (one step for function calls), $\Delta T_M = 1$ (one step for function calls), and $\Delta T_S = 1 \times N_p$. Hence

$$T_{\mathrm{GA}} = \left[2L \times N_d + (3 + N_p)\right] \times N_g. \tag{14}$$

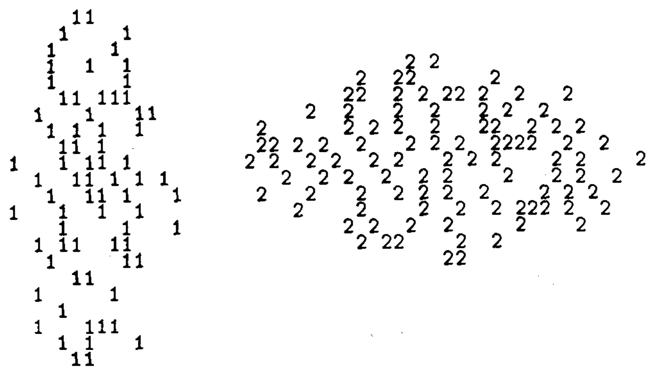
Now,

$$T_{\mathrm{BP}} - T_{\mathrm{GA}} = 2LN_d(N_i - N_g) + \left[3(L-1)N_dN_i - (3 + N_p)N_g\right]. \tag{15}$$

We have $L \geqslant 2$. Let us assume that $N_i = N_g$ (i.e., the number of iterations required in backpropagation is equal to that required in the genetic algorithm to obtain a desired output). Therefore, from (15),

$$T_{\mathrm{BP}} - T_{\mathrm{GA}} \geqslant \left[3N_dN_i - (3 + N_p)N_i\right].$$

Here, $T_{\mathrm{BP}} > T_{\mathrm{GA}}$ if $N_d \geqslant N_p/3 + 1$.


(a) A, linearly separable

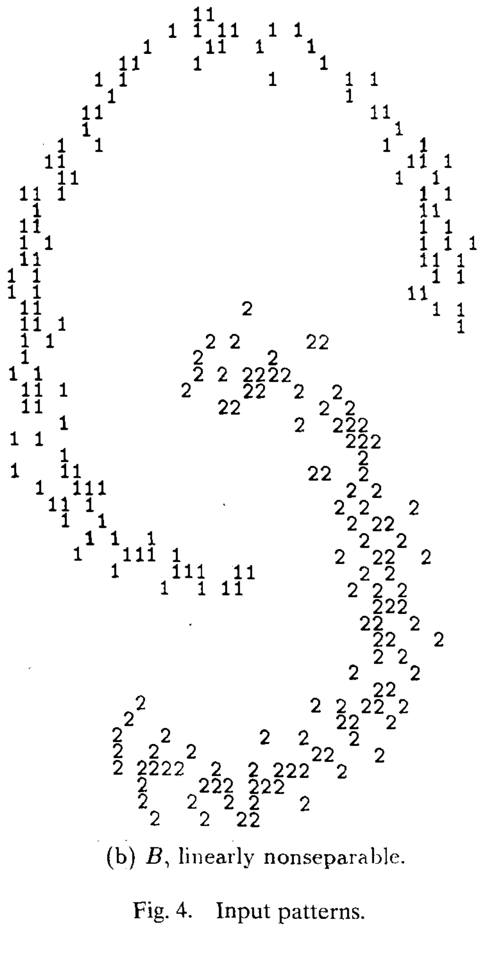

(b) B, linearly nonseparable.

Fig. 4. Input patterns.

Usually, $N_d > N_p$, so we can conclude that for the same number of iterations, $T_{BP} > T_{GA}$. Again the experiment shows that N_i is always greater than N_g (Figures 8–11) for a desired output. This means the first part $(2LN_d(N_i - N_g))$ of (15) is greater than zero, thereby making the claim more valid.

6. IMPLEMENTATION AND RESULTS

To demonstrate the effectiveness of the proposed algorithm for selecting the appropriate weights of an MLP, we considered both linearly separable and nonseparable patterns having two features taken from two classes. Two such pattern sets A and B (Figure 4(a)–(b)), artificially generated, consisting of 200 pattern points have been considered here for simulation. Set A is linearly separable and set B is linearly nonseparable and involves nonconvex decision regions. The size of the training sets is considered to be 40% of all classes. The existing backpropagation technique is also implemented for pattern set B for its comparison with the proposed method.

The two-layer network consisting of two input nodes, two output nodes, and four links is considered to classify the linearly separable pattern set A. There are six parameters (four are associated with the four links and two for the biases of two output nodes). Therefore, strings of length 60

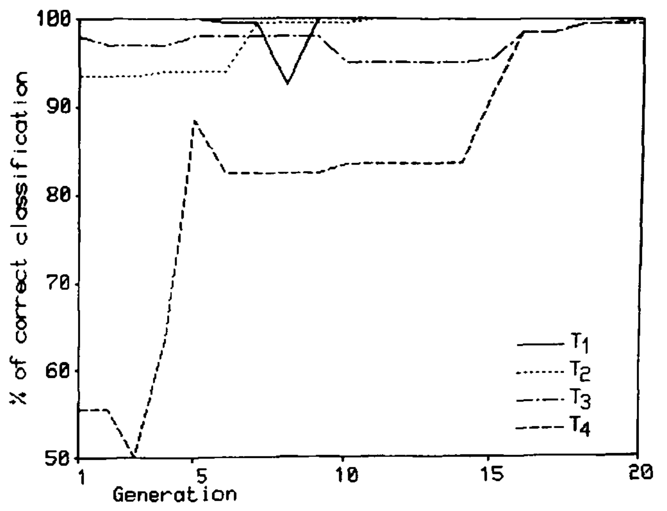


Fig. 5. Performance of the proposed algorithm on pattern set A for four training data sets.

(substring of length 10 corresponds to each parameter) are considered here for the chromosomal representation of the solution. The sigmoidal function of (3) is used here as the activation function. We have used the error function given in (8) and the linear selection process [(9)].

Figure 5 shows the performance of the proposed algorithm for pattern set A. Four different training data sets (T_1, T_2, T_3, T_4) of same size have been selected randomly to learn the network parameters. For each training set, a separate initial population of size 50 (generated randomly) has been considered. In each case, it was found that the algorithm correctly classified all the patterns of the set. Only a few generations (10 to 20) were required to classify the patterns. The algorithm has also been tested with a population of size 20 and, as expected, it is found to take more generations for correct classification (Figure 6). Figure 7 depicts the gradual convergence of the algorithm (for population size 20), i.e., how the error of classification reduces with generation.

For pattern set B (with linearly nonseparable classes), a network having three layers (as shown in Figure 2) is used. Here we have demonstrated the effect of training set and initial population on the classification performance. Figure 8 shows the results obtained using four different training sets for a fixed initial population P_1. In each case, the performance of the proposed technique is found to be satisfactory after 50 generations. The

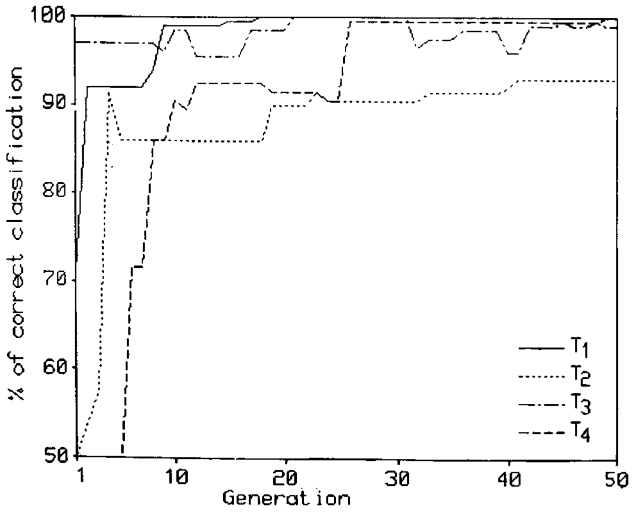


Fig. 6. Performance of the proposed algorithm, when the population size is 20, on A for four training data sets.

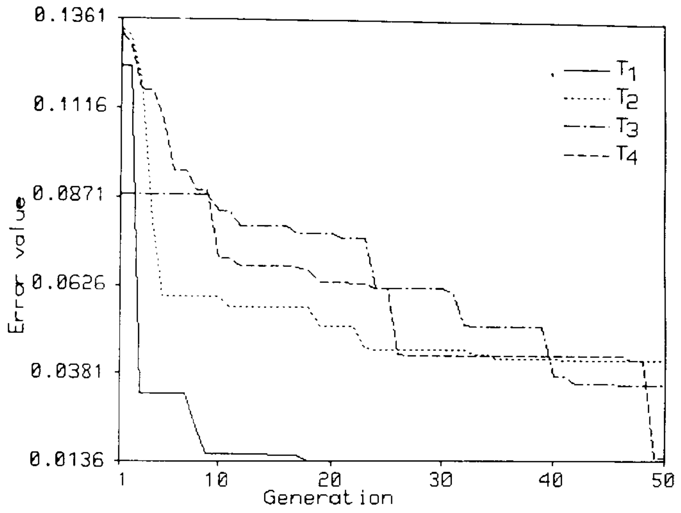


Fig. 7. Gradual convergence of the error of classification with respect to generation.

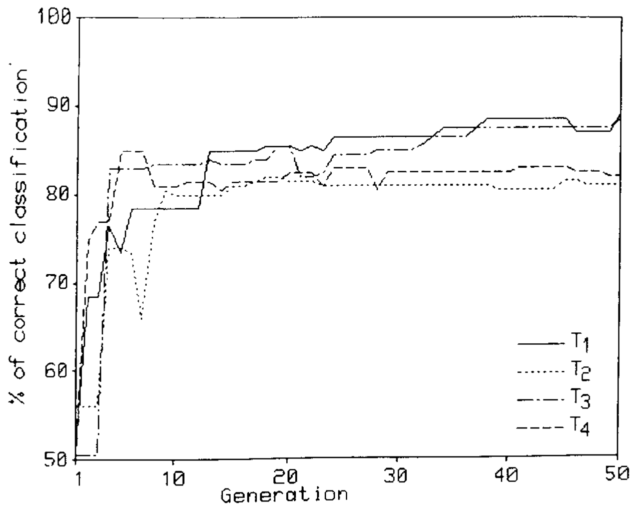


Fig. 8. Performance of the proposed algorithm on B for four training data sets and a fixed initial population P_1.

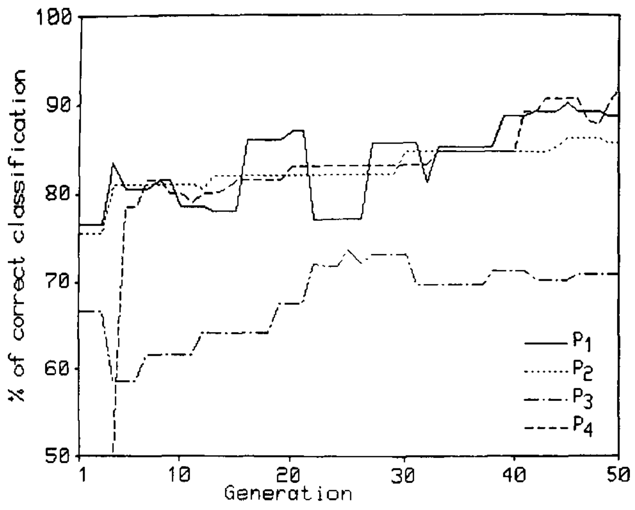


Fig. 9. Performance of the proposed algorithm on B for four initial populations and a fixed training data set T_4.

results obtained for four different initial populations and a fixed training data set T_4 are shown in Figure 9. Note from Figures 8 and 9 that the algorithm performs well in any environment except for the population P_3. Though the performance in the case of P_3 was better in the early generations as compared to many others, ultimately it failed to improve the performance. This may possibly be due to improper crossover and mutation operations. As in the case of A, the number of generations required for desired performance increases with decreasing population size.

The existing backpropagation technique is implemented on the same network for comparison, keeping T_1, T_2, T_3, and T_4 the same as before. This investigation has two parts. In one part (Figure 10), the initial approximation of weights I_1 is kept fixed and in the other (Figure 11), training set T_4 is kept fixed. It is seen that the network needs a huge number of iterations to provide results comparable (except I_2 and I_3 in Figure 11) to that of the proposed algorithm. For I_2 and I_3, the system is unable to improve even after about 200 iterations, thereby demonstrating the dependency of the backpropagation technique on the initial approximation of weights.

In the final part of the investigation, we have investigated the effect on the nonlinear selection process on system performance for pattern set B.

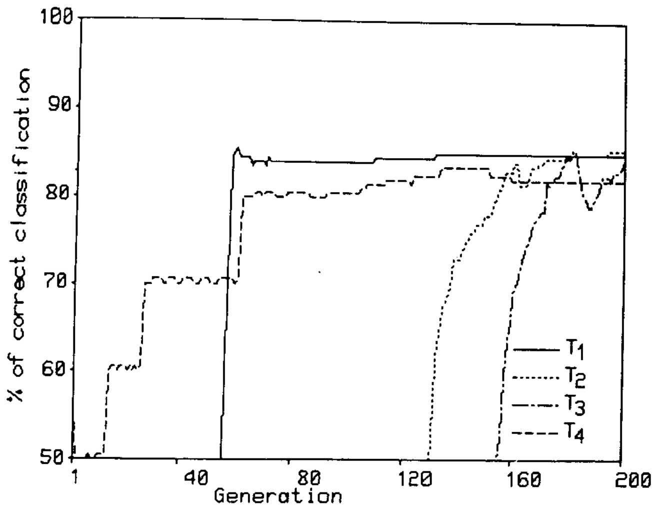


Fig. 10. Output performance of backpropagation technique for a fixed initial approximation I_1 on B for four training data sets.

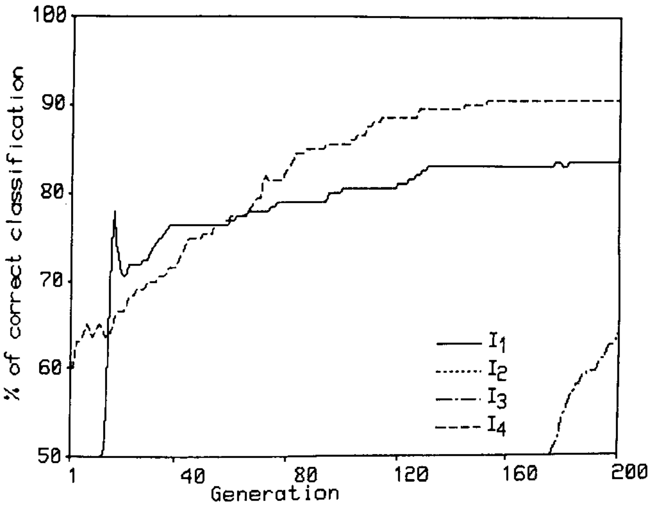


Fig. 11. Output performance of backpropagation technique for a fixed training data set T_4 on B for four initial approximations.

As an illustration, we have shown in Figure 12 such an effect when different initial populations P_1, P_2, P_3, and P_4 (as described for Figure 9) were considered for a fixed training set T_4. With the nonlinear selection method [(10)], the algorithm is able to produce comparable results (see Figure 12) even with the initial population P_3 for which the linear selection process could not provide better result (Figure 9). Note that the rate of correct classification with respect to generation is higher (except P_4) than that of the previous experiment. For P_4, although its initial classification score was much higher than that in Figure 9, the final output is not so high comparatively. It was revealed under investigation that the populations in early generations for P_4 did not have homogeneity among the potential strings as compared to the cases P_1, P_2, and P_3. The application of nonlinear selection therefore rejected some of the important genetic information for P_4, thus slightly reducing the scope of further improvement.

7. CONCLUSIONS

The effectiveness of GAs in selecting the optimum set of parameters of a multilayer perceptron for pattern recognition problem is demonstrated. The classification method is found to be suitable not only for providing an

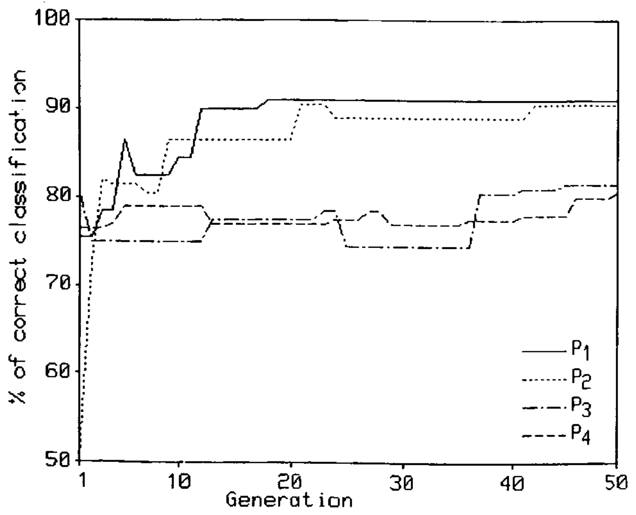


Fig. 12. Effect of nonlinear selection on B for four initial populations and a fixed training data set T_4.

optimal (global) set of weights of the neural network, but also for reducing the computational time (because of the avoidance of the backpropagation task). This algorithm is implemented for a two-class linearly separable/nonseparable pattern for different training sets, population sizes, and initial populations. The algorithm is seen to be robust (as in [4–6]) and the results obtained are satisfactory.

The number of generations required for a desired performance is seen to decrease with increasing population size. The time of computation for parallel implementation of the proposed algorithm and the backpropagation algorithm has been determined. It has been found that the backpropagation technique requires a larger number of iterations to produce results comparable to those of the proposed algorithm. Unlike the backpropagation technique, the proposed method does not depend on the sequence of the training data. The incorporation of the weighted average as a fitness function is found to take care of the unequal occurrence of classes in positioning decision boundaries. The nonlinear selection process, which increases the genetic homogeneity of the population, is found to enhance the search speed. However, this needs further investigation for determining the appropriate selection procedure.

The proposed algorithm determines the optimum parameter set, not the individual parameters, in selecting an appropriate decision boundary. Although GAs consider a large search space (which increases the possibility of getting better results), they require, in practice, only a smaller number of points to achieve the result. The domains of the parameters here are continuous. Therefore, to obtain a more accurate solution, one needs to increase the length of the strings, though it will increase the computational time. The proposed approach of selecting an optimum set of parameters can be used for stabilization of any other connectionist model.

The authors gratefully acknowledge Dr M. K. Kundu and A. Ghosh for helpful discussions.

REFERENCES

1. D. E. Rumelhart, J. McClelland, and P.D.P. Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, Cambridge, MA, 1986.
2. D. E. Goldberg, *Genetic Algorithms: Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
3. L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 198

4. S. Bornholdt and D. Graudenz, General asymptotic and neural networks and structure design by genetic algorithms, *Neural Networks* 5:327–334 (1992).
5. H. Muhlenbein, Limitations of multi-layer perceptron networks—steps towards genetic neural networks, *Parallel Comput.* 14:249–260 (1990).
6. D. Whitley, T. Starkweather, and C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, *Parallel Comput.* 14:347–361 (1990).
7. W. Siedlecki and I. Sklansky, A note on genetic algorithms for large-scale feature selection, *Pattern Recognition Lett.* 10:335–347 (1989).
8. C. A. Ankerbrandt, B. P. Unckles, and F. E. Petry, Scene recognition using genetic algorithms with semantic nets, *Pattern Recognition Lett.* 11:285–293 (1990).
9. D. Bhandari, S. K. Pal, and M. K. Kundu, Image enhancement incorporating fuzzy fitness function in genetic algorithms, in: *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'93)*, IEEE, San Francisco, 1993, pp. 1408–1413.
10. S. K. Pal and S. Mitra, Fuzzy versions of Kohonen's net and mlp based classification: performance evaluation for certain non-convex decision regions, *Inform. Sci.* To appear.
11. S. K. Pal and D. D. Majumder, *Fuzzy Mathematical Approach to Pattern Recognition*, Wiley (Halsted Press), New York, 1986.

Received 13 January 1993; revised 14 June 1993; accepted 15 October 1993