# Evolving Multilayer Neural Networks using Permutation free Encoding Technique

**Anupam Das and Saeed Muhammad Abdullah**
Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.
Email: anupamdas@cse.buet.ac.bd, saeed.siam@gmail.com.

**Abstract**—*In this paper we present an evolutionary system using genetic algorithm (GA) for evolving artificial neural networks (ANNs). Existing genetic algorithms for evolving ANNs suffer from the permutation problem as a result of recombination. Here we propose a novel encoding scheme for representing ANNs which avoids the permutation problem while efficiently evolving multilayer ANN architectures. The evolutionary system has been implemented and tested on a number of benchmark problems in machine learning and neural networks. Experimental results suggest that the system shows superiority in performance, in most of the cases.*

**Keywords:** Permutation problem, genetic algorithm (GA), recombination, artificial neural network(ANN).

## 1. Introduction

Evolution has gained popularity as an alternative to classical search and optimization methods due to its natural adaptiveness and ANNs have always provided a practical way of representing a function such as a classifier when the input data is complex or noisy. The typical approach is to use a fixed network architecture with random connection weights, and then use a backpropagation (BP) algorithm that adjusts the connection weights until the desired accuracy is achieved. Since in this approach the architecture is constrained, there is no opportunity for the learning algorithm to find a better architecture for the problem. But the information processing capability of an ANN is determined by its architecture. So for different applications the architecture should be different.

However the problem of designing a near optimal ANN architecture for an application remains unresolved. Design of a near optimal ANN architecture can be formulated as a search problem in the architecture space where each point represents an architecture. Based on some optimal criteria, the best performing network has to be selected which is equivalent to finding the highest point on the search space. This is where evolutionary algorithms work better than others because it is less likely to get stuck in a local maxima due to its strong tendency towards exploration.

This paper is an extension of our previous work (PETENN) [1] which dealt with only single hidden layer architectures and was thus, not applicable to all non-linear problems. Here we describes an evolutionary system, i.e., EMNNPET (Evolving Multilayer Neural Networks using Permutation free Encoding Technique), for evolving multilayer feedforward ANNs which can be applied to any kind of non-linear problem. The algorithm combines architectural evolution with weight learning.

## 2. EMNNPET

EMNNPET uses a genetic algorithm for evolving ANNs. The encoding scheme is designed to avoid the permutation problem and make the whole process more efficient. The recombination operator may be normal or mixed as described later. Such operators ensure automation in the addition or deletion of nodes and/or links from the parents. As a result, the whole process is free from user interaction. Mutation operators are also applied randomly with a very small probability.

The steps of the algorithm are as follows:

1) Generate an initial population of N networks at random each of which is free from permutation problem. Nodes and connections are uniformly generated at random within a certain range.Weights are also generated uniformly at random within a certain range.
2) Train each network partially for certain number of epochs using BP.
3) If *stopping criteria*(described later on) is met, then stop the process and identify the best network. Otherwise go to step 4.
4) Do a *random shuffle* on current population and pair them up. The recombination operator and mutation operator is applied on each pair to obtain two offspring. Ignore the offspring that introduces permutation problem.
5) The best N individuals are transferred to the next generation (a variant is to use “Elitist replacement” scheme where local competition is held instead of global competition, and thus discouraging a greedy approach).
6) Go to step 2.

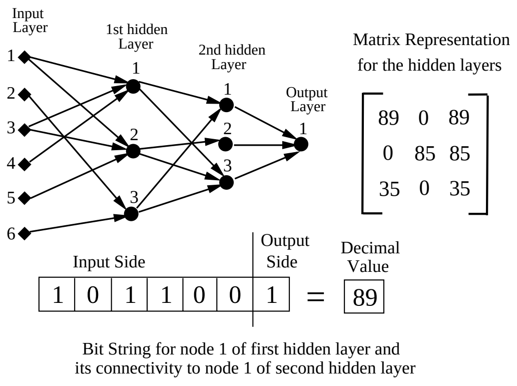

Input Layer
1st hidden Layer
2nd hidden Layer
Output Layer
Matrix Representation for the hidden layers
89 0 89
0 85 85
35 0 35
Input Side
Output Side
Decimal Value
1 0 1 1 0 0 1 = 89
Bit String for node 1 of first hidden layer and its connectivity to node 1 of second hidden layer

Fig. 1: An ANN and its genotype in the proposed encoding scheme.

## 2.1 Encoding Scheme

The encoding scheme is direct in nature where architecture is encoded. The proposed encoding scheme ensures that neural networks with behaviorally equivalent structure will produce similar genotypic representations. In this paper we consider only three-layer perceptrons. So there are four layers of nodes: input layer, first hidden layer, second hidden layer and output layer. The input and output layers are fixed for a particular problem. Only the two hidden layers evolve. The two hidden layers are represented by a m$*$n matrix *M[ ]*, where m (row) represents the number of nodes in the first layer and n (column) represents the number of nodes in the second layer. Each element of the matrix represents a decimal value defined by a bit string corresponding to the connections with the input and output nodes. One bit is needed for each of the input and output nodes. The value '**1**' means a connection exists and '**0**' means no connection.

The matrix element *M[i , j]* can be-

- **Non-zero positive** value which means that there is a connection between the *i-th* node of the first hidden layer and the *j-th* node of the second hidden layer.
- **Zero** value which means that there is a no connection between the *i-th* node of the first hidden layer and the *j-th* node of the second hidden layer.

So for a non-zero value *M[i , j]* represents a bit string corresponding to not only the information about the connections with the input nodes and the *i-th* node of the first hidden layer, but also the connections with the *j-th* node of the second hidden layer and the output nodes. Fig. 1 shows the encoding technique for an ANN.

## 2.2 Fitness Evaluation

The fitness of each individual is solely determined by the inverse of an error function E suggested by

$$E = 100\frac{O_{max} - O_{min}}{Tn}\sum_{t=1}^{T}\sum_{i=1}^{n}\left(Y\left(i,t\right) - Z\left(i,t\right)\right)^2 \ . \quad (1)$$

Here, $O_{max}$ = maximum value of output coefficients, $O_{min}$ = minimum value of output coefficients, n = number of output nodes, T = number of patterns, $Y\left(i,t\right)$ = actual outputs of nodes, $Z\left(i,t\right)$ = desired outputs of nodes. The error measure is less dependent on the size of the validation set and the number of output nodes.

## 2.3 Parent Selection

The first step of recombination is selection of parents. Here uniform selection is used i.e., each individual has exactly the same probability of being selected as a parent. To implement this uniform selection the population is randomly shuffled and parents are selected by taking pairs of individuals.

## 2.4 Recombination Operator

Two types of recombination operator are applied: normal and mixed. The recombination is applied to the matrix representation of the two hidden layers. Here the column number of the two matrices must be equal. So, incase of unequal column number a **zero column vector** (or vectors) $[0\ 0\ 0\ ...]'$ (row number same as previous) is appended to the rightmost end of the smaller matrix.

### 2.4.1 Normal Recombination Operator

In this scheme the number of crossover points are random and limited to a highest value. Then alternate parts are taken from the parents to build up the offspring. Fig. 2 shows how child-1 and child-2 are produced from parent-1 and parent-2 using normal recombination operator.

### 2.4.2 Mixed Recombination Operator

In this scheme, the genotypes(hidden nodes) of the two parents are mixed by concatenation. Then a random crossover point is taken and the two portions correspond to the two offspring. Fig. 3 shows how child-1 and child-2 are produced using mixed recombination operator.

## 2.5 Evolution of Network Architecture

The recombination operators ensure automation in the following forms:

- Connections (links) addition / deletion
- Nodes addition / deletion

The following sections describe the scenarios with examples.

### 2.5.1 Connection Addition / Deletion

In the proposed encoding scheme, the nonzero decimal values correspond to how 1st layer hidden nodes are connected to the inputs and how the 2nd layer hidden nodes are connected to the outputs. So any change in the encoded value implies alterations in those connections. Such an example is shown in Fig. 4 where the genotypes of the children are different to those of their parents and this denotes connection add/delete as described above.

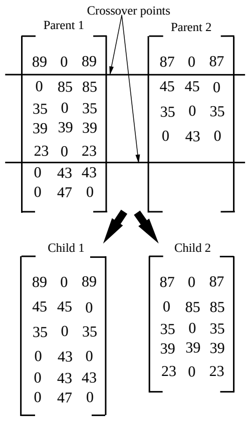


Fig. 2: Schematic representation of normal recombination.

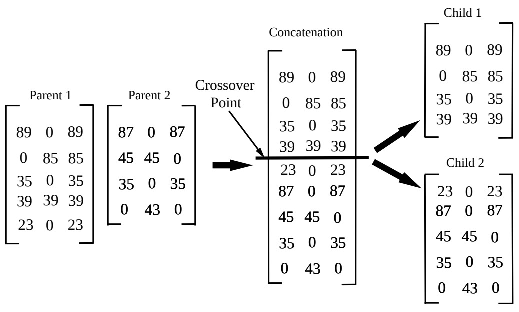


Fig. 3: Schematic representation of mixed recombination.

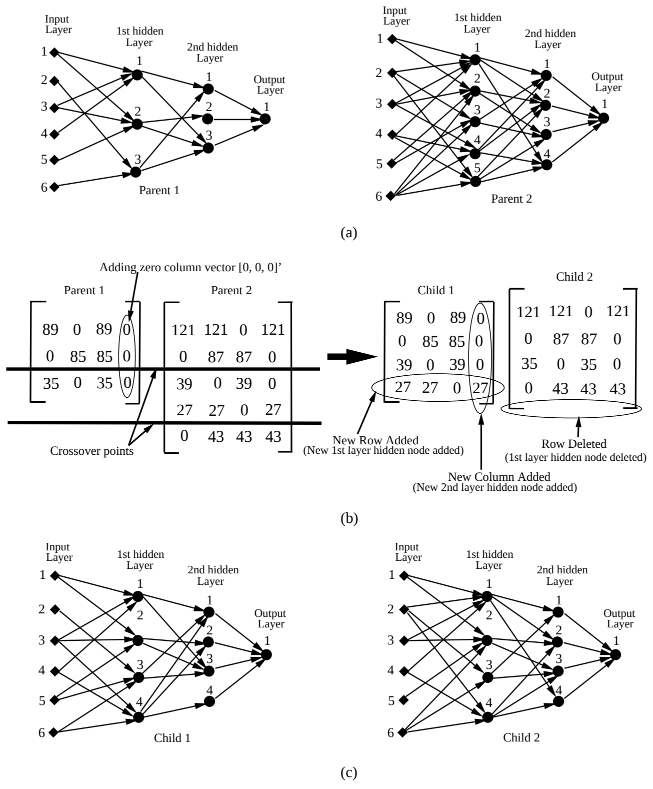


Fig. 4: (a) The two parents used in recombination (b) Recombination using multiple crossover points. This shows how addition and deletion of connections and nodes takes place (c) The two offspring produced.

### 2.5.2 Node Addition / Deletion

Nodes will be added or deleted from the parent ANN if the following conditions are met:

- The parents have unequal number of nodes in the 1st and/or 2nd hidden layer.
- Multiple crossover points are used.

This is shown in Fig. 4 where parent-1 has a total of six hidden nodes and parent-2 has a total of nine hidden nodes. Here two crossover points are used. Now after recombination both child have a total of eight hidden nodes. So the number of hidden nodes in the children is different to that in the parents and hence node add/delete is possible in such type of recombination.

## 2.6 Mutation

The mutation operator may be applied on the offspring after recombination. The mutation operators are applied with very small probabilities. The following mutation operators can be applied on the offspring: Firstly, a new hidden node may be added to the network. The links of this node with the input and output nodes are randomly generated. The weights of the links are also random within a certain range. Secondly, a hidden node of the network may be selected randomly and then deleted from the network. The probability of deleting a node is more than that of adding it since we want to reduce the size of the network if possible. Thirdly, the network may be mutated by deletion of a link. Again the link is chosen randomly. And fourthly, the weight of a randomly selected link may be changed randomly within a certain range.

## 2.7 Parent Replacement

For a population of N individuals after recombination and mutation the total number of individuals can become at most 2N (given that no permutation occurs). Among these individuals the best N individuals are transferred to the next generation. As a variant we can also use “Elitist replacement” scheme where a competition is held between the two offspring and their two parents, and the two winners are transferred to the next population, thus discouraging a greedy approach in the replacement of individuals.

## 2.8 Permutation Problem

The main aspect of the proposed encoding scheme is the solution to permutation problem. The following section describes the concept of permutation problem and the next one illustrates the solution with example.

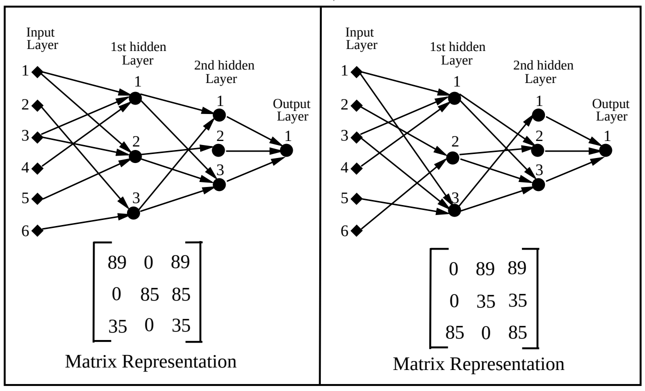


Fig. 5: Example of permutation problem. (a) An ANN and its genotypic representation. (b) Another ANN identical to that given in (a) with different genotypic representation.

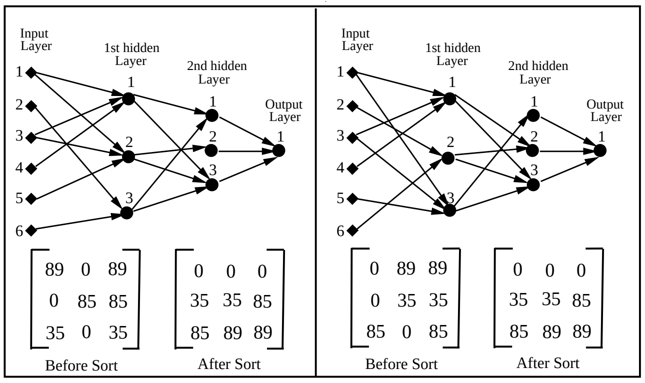


Fig. 6: Similar ANNs with different orientation produce similar genotypes. (a) An ANN and its genotypic representation. (b) Another ANN identical to that given in (a) with same genotypic representation after sorting.

### 2.8.1 Concept of the Permutation Problem

The evolution of ANN architectures in general suffers from the permutation problem [2], [3]. It is caused by the many to one mapping from genotypes to phenotypes since two ANNs which order their hidden nodes differently may have different genotypes but are behaviorally (i.e., phenotypically) equivalent. With a purely feed-forward net, there is no significance to the order of the hidden units. This means that the genetic coding is redundant. This problem is depicted in Fig. 5 where a direct encoding scheme is used. Here two networks are shown which are identical except that their nodes in the hidden layer are in different order. Permutation problem makes the evolution inefficient as the population may have several members that are identical in structure but different in genotypic representation. It also complicates the task for crossover operators to produce highly fit offspring.

### 2.8.2 Solution to Permutation Problem

It is obvious that if two hidden nodes have same bit patterns and hence same values then they have similar connections. Now consider two structurally similar ANNs with their hidden nodes in different order. The matrix representation for their hidden layers will also be similar but in different orientation. In our encoding scheme, the matrix elements are sorted so that *M[1 , 1]* contains the ***minimum value*** and *M[row-number , column-number]* contains the ***maximum value***. Then these two ANNs will produce one-to-one mapping from genotypes to phenotypes, thus solving the permutation problem.

For example, the two ANNs shown in Fig. 6 are similar in structure but their hidden nodes are in different order. Thus their directly encoded matrix representation (initial genotype) are also different. But in the proposed encoded scheme the genotypes are sorted and finally the two ANNs produce same genotype.

Table 1: Characteristics of the Data sets

| Data set | Number of examples | Number of input attributes | Number of output classes |
|---|---|---|---|
| Breast cancer | 699 | 9 | 2 |
| Diabetes | 768 | 8 | 2 |
| Heart disease | 303 | 35 | 2 |
| Thyroid | 7200 | 21 | 3 |

## 3. Experimental Studies

This section evaluates EMNNPET's performance on some of the well known benchmark problems. These problems have been the subject of many studies in NNs and machine learning.

### 3.1 The Medical Diagnosis Problems

EMNNPET was applied to four classical classification problems in the medical domain i.e., the breast cancer problem, the diabetes problem, the heart disease problem, and the thyroid problem. All the datasets were obtained from the University of California, Irvine (UCI) machine learning benchmark repository, available at http://ftp.ics.uci.edu/pub/machine-learning-databases/. Detailed descriptions of data sets are also available at this site. The characteristics of the data sets are summarized in Table 1.

### 3.2 Experimental Setup

All the data sets used have been partitioned into three sets: a training set, a validation set, and a testing set. In most case, the partitioning is 50%-25%-25% except for the thyroid problem since it contains huge amount of examples. The training set is used to train ANN by backpropagation algorithm; the validation set is used to evaluate the fitness and checking over fitness of an ANN and the testing set is used to evaluate the performance of the system. In the following experiments, each data set was partitioned as shown in Table 2.

Table 2: Data set partitions

| Data set | Number of Training examples | Number of Validation examples | Number of Test examples |
|---|---|---|---|
| Breast cancer | 350 | 175 | 174 |
| Diabetes | 384 | 192 | 192 |
| Heart disease | 152 | 75 | 76 |
| Thyroid | 2514 | 1288 | 3428 |

It, however, should be kept in mind that such partitions do not represent the optimal ones [4]. For each of the problems, the input attributes are discrete on a scale 0 - 1 (real) and output attribute is binary valued. The output attributes of all the problems were encoded using a 1-of-m output representation for m classes.

The initial population consists of a random number of individuals. Each individual is an ANN of two hidden layers as it is sufficient to solve all non-linear problems [5]. Initially, the number of nodes in a hidden layer is uniformly chosen between *minimum_value* =(input_node+output_node)$*0.5$ and *maximum_value* =(input_node+output_node)$*1.5$. Initially, connections between the two hidden layers and the 1st hidden layer and input layer are randomly created. The output layer is fully connected. Initial weights are assigned between -1.0 to 1.0.

Most of the parameters used here are not meant to be optimal. But, these are set after certain phases of tuning during several runs. The activation function used is

$$sigmoid = \frac{1}{(1+e^{-x})} . \tag{2}$$

Learning rate of the back-propagation algorithm is set to 0.5. No momentum factor is used. The number of epochs is set to 10-20 for initial training and 5-10 for partial training. The *stopping criterion* for the algorithm is based on a predefined *threshold values*. If the average error does not fall below the threshold after a maximum number of iterations (300 or 500), the evolution is stopped at that point.

## 3.3 Experimental Results

Tables 3–4 show the results of EMNNPET over 30 independent runs on the four different problems. The error in the tables refers to the error defined by fitness function E (1). The **error rate** refers to the percentage (%) of wrong classifications produced by the evolved ANN's.

Table 3 summarizes the average, standard deviations (SD), minimum, maximum number of hidden nodes, connections and generations for each set of problems. From the table we can see that the number of generations required to meet the stopping criteria is quite low.

Table 4 shows the mean, standard deviation, minimum and maximum value of validation and testing errors for the four problems. Most of the errors are small and are competitive with the errors provided by existing other algorithms.

In order to observe the evolutionary process in EMNNPET, following graphs Figs. 7–8 show the evolution of the error and number of cumulative permutations detected over 30 runs for the four medical diagnosis problems.

The evolutionary process is quite interesting. The number of connections in the ANN decreases and increases through out the process signifying that the recombination operator works as a exploration operator to find the best suitable network. Similarly as the generation proceeds the amount of error decreases significantly which implies that the algorithm in moving towards the best structure for the corresponding problem. As we can see in the figures that error drops quite quickly to the lowest value signifying the effectiveness of the recombination and mutation process. Moreover we can see from Fig. 8 that the number of permutations detected in each generation is quite significant and hence the steep rise in the curves.

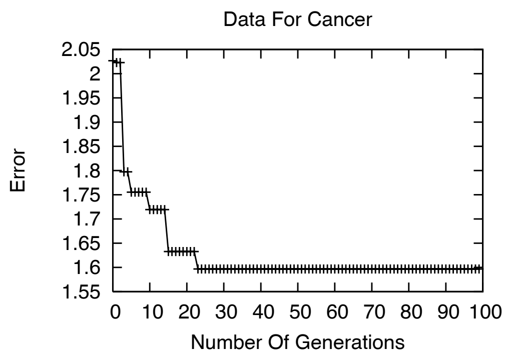
## 3.4 Comparison with Other Works

The aim of this paper is not just, to compare this algorithm with other algorithms but to establish a permutation free encoding scheme that could be used in existing algorithms. Moreover, direct comparison with other evolutionary approaches of designing ANN is very difficult due to the lack of such results. Instead, the best available results, regardless of whether the algorithm used was an evolutionary, a BP or a statistical approach, are used in the comparison.

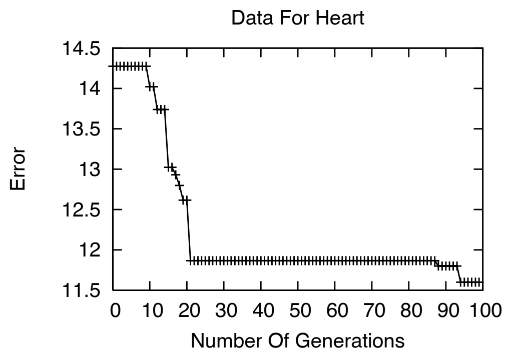
This section compares experimental results of EMNNPET with those of FNNCA [6], Prechelt's hand designed best NN (HDANNs) [7], EPNet [8], Bennet and Mangasarian's [9] MSM1, Schiffmann [10], CNNE [4], acasper [11] , AMGA [12], COVNET [13], REANN [14], MCA [15]. EMNNPET has been compared to other works in terms of Average test error rate (TER) over 30 runs, for each of the problems in the following tables. Since not all algorithms have been tested on the same set of problems, therefore only those that are available for a given problem have been compared to EMNNPET.

Table 3: Architecture of Evolved Neural Networks

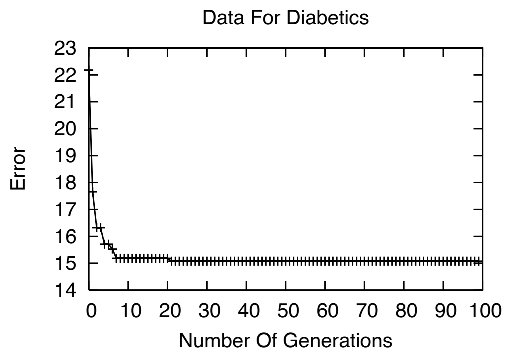
|  | Measurement | Number of connections | Number of hidden nodes | Number of generations |
|---|---|---|---|---|
| Breast Cancer Data Set | Mean | 86.6667 | 14.1 | 4.7755 |
|  | SD | 2.3094 | 1.3152 | 3.3619 |
|  | Min | 84 | 11 | 1 |
|  | Max | 88 | 16 | 23 |
| Heart Disease Data Set | Mean | 472.90 | 40.21 | 86.04 |
|  | SD | 55.87 | 14.02 | 62.7336 |
|  | Min | 309 | 23 | 23 |
|  | Max | 676 | 116 | 300 |
| Diabetes Data Set | Mean | 95.5 | 14.6 | 5.9591 |
|  | SD | 8.0436 | 1.4142 | 1.8925 |
|  | Min | 84 | 11 | 3 |
|  | Max | 104 | 17 | 12 |
| Thyroid Data Set | Mean | 304.25 | 23.5 | 35.67 |
|  | SD | 11.19 | 5.29 | 3.08 |
|  | Min | 277 | 14 | 32 |
|  | Max | 391 | 29 | 41 |



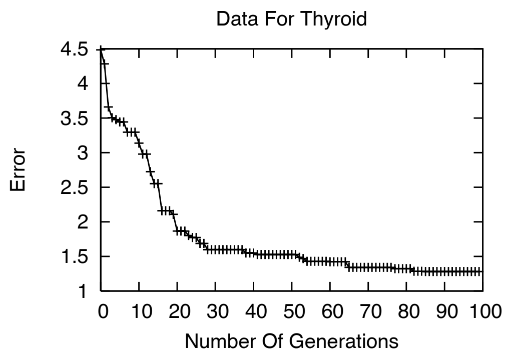
(a)



(b)

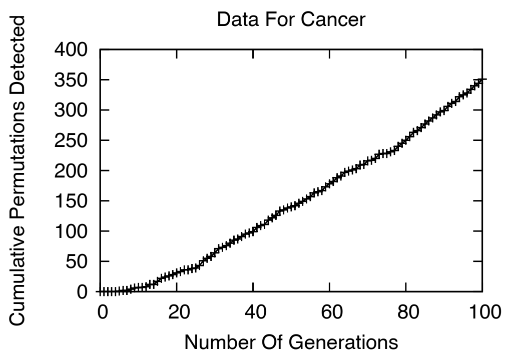


(c)

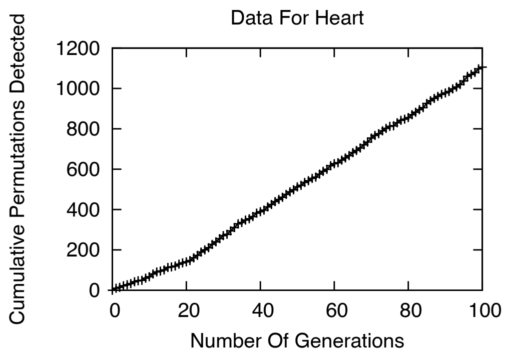


(d)

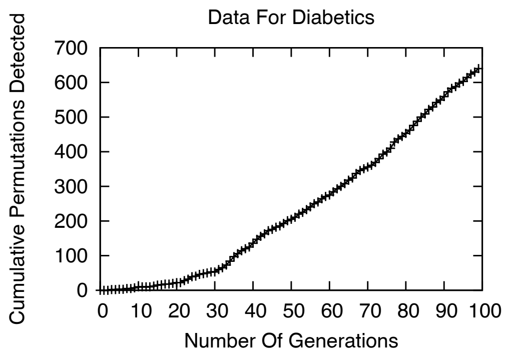
Fig. 7: Evolution of ANN's error for (a) breast cancer problem (b) heart disease problem (c) diabetes problem (d)thyroid problem.



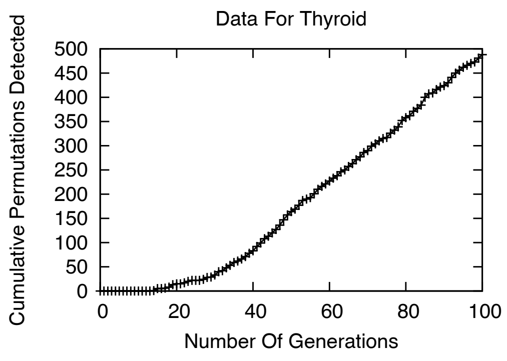
(a)



(b)



(c)



(d)

Fig. 8: Permutations detected in the evolution of ANNs (a) breast cancer problem (b) heart disease problem (c) diabetes problem (d)thyroid problem.

Table 4: Accuracies of Evolved Artificial Neural Networks

| | Measure -ment | Validation Set Error | Validation Set Error rate | Test Set Error | Test Set Error rate |
|---|---|---|---|---|---|
| Breast Cancer Data Set | Mean | 1.7533 | 2.3657 | 0.9989 | 1.0344 |
| | SD | 0.0373 | 0.4427 | 0.1539 | 0.5010 |
| | Min | 1.6280 | 1.7142 | 0.6026 | 0.0 |
| | Max | 1.7975 | 3.4285 | 1.3735 | 2.2988 |
| Heart Disease Data Set | Mean | 11.4932 | 12.6315 | 5.7773 | 6.9866 |
| | SD | 0.2409 | 1.3157 | 1.5328 | 2.2400 |
| | Min | 10.6745 | 9.2105 | 1.6981 | 1.3333 |
| | Max | 11.7702 | 15.7895 | 10.7219 | 12 |
| Diabetes Data Set | Mean | 15.5915 | 21.7395 | 16.4768 | 24.4895 |
| | SD | 0.1438 | 1.2272 | 0.3977 | 1.8735 |
| | Min | 15.18 | 20.8333 | 15.6787 | 19.2708 |
| | Max | 15.7481 | 24.4792 | 17.1544 | 29.1667 |
| Thyroid Data Set | Mean | 0.8250 | 1.6111 | 0.8795 | 1.5185 |
| | SD | 0.0381 | 0.2525 | 0.1436 | 0.2143 |
| | Min | 0.7750 | 1.2777 | 0.6775 | 1.2222 |
| | Max | 0.8676 | 1.8888 | 0.9995 | 1.7222 |

Table 5: Comparing EMNNPET in terms of Average TER with other algorithms for the breast cancer problem

| | EMNNPET | FNNCA | HDANNs | EPNet | CNNE | AMGA | REANN |
|---|---|---|---|---|---|---|---|
| TER | 1.03 | 1.45 | 1.15 | 1.37 | 1.3 | 1.3 | 3.72 |

Table 6: Comparing EMNNPET in terms of Average TER with other algorithms for the heart disease problem

| | EMNNPET | MSM1 | HDANNs | EPNet | CNNE | AGMA | COVNET | MCA |
|---|---|---|---|---|---|---|---|---|
| TER | 6.98 | 16.53 | 14.78 | 16.7 | 13.4 | 18.87 | 14.26 | 18.91 |

Table 7: Comparing EMNNPET in terms of Average TER with other algorithms for the diabetes problem

| | EMNNPET | HDANNs | EPNet | CNNE | acasper | AGMA | REANN | MCA |
|---|---|---|---|---|---|---|---|---|
| TER | 24.4 | 21.35 | 22.4 | 19.8 | 20.31 | 21.97 | 23.44 | 20.99 |

Table 8: Comparing EMNNPET in terms of Average TER with other algorithms for the thyroid problem

| | EMNNPET | Schiffmann | HDANNs | EPNet | AGMA |
|---|---|---|---|---|---|
| TER | 1.51 | 2.5 | 1.28 | 1.63 | 2.44 |

Table 5 shows that EMNNPET achieves the best result of 1.03% TER. Table 6 shows that EMNNPET has the lowest average testing error rate among all other algorithms for the heart disease problem. It outperforms the rest by quite a significant margin. This due to recombination operators which imposes very few constraints on the evolution of ANNs. Table 7 shows that EMNNPET performed worst and CNNE performed best among the algorithms for diabetes problem. But CNNE is not a genetic approach but rather an evolutionary ensemble approach. According to Table 8, EMNNPET performs the second best among all the other algorithms for the thyroid problem. This superiority in performance is due to the exploration capability of the recombination operators.

## 4. Conclusion

In this paper we have provided an innovative encoding scheme for the representation of ANNs that can solve the permutation problem. A mixed recombination operator has also been proposed. Another feature is the automation of addition or deletion of nodes or links during the recombination process. Our algorithm, EMNNPET has been tested on a number of benchmark problems and it has shown better results compared to other algorithms in most of the cases. EMNNPET imposes very few constraints on feasible ANN architectures; thus explores a huge search space of different ANNs.

As future work, we want to concentrate on the recombination process because it can be done in different ways and the effect of those methods can give better results. Specially we are interested in minimizing the network size without compromising much of the ANN's performance.

## References

[1] A. Das, S. Hossain, S. M. Abdullah, and R. U. Islam, "Permutation free encoding technique for evolving neural networks," in *Proceedings of the 5th international symposium on neural networks, ISNN 2008*, vol. 5263/2008, Sep. 2008, pp. 255–265.

[2] R. K. Belew, J. McInerney, and N. N. Schraudolf, "Evolving networks: Using the genetic algorithm with connectionist learning," in *Proceedings of the Workshop on Artificial Life*, 1992, pp. 511–547.

[3] P. J. B. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *Proc. COGANN Workshop*, 1992, pp. 108–122.

[4] M. Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 820–834, Jul. 2003.

[5] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.

[6] R. Setiono and L. C. K. Hui, "Use of a quasinewton method in a feed forward neural-network construction algorithm," *IEEE Transactions on Neural Networks*, vol. 6, pp. 273–277, 1995.

[7] L. Prechelt, "Proben1-a set of neural network benchmark problems and bench-marking rules," Fakultat fur Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, Sept. 1994.

[8] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, May 1997.

[9] K. P. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," vol. 1, pp. 23–34, 1992.

[10] W. Schiffmann, M. Joost, and R. Werner, "Synthesis and performance analysis of multilayer neural network architectures," Univ. Koblenz, Inst. fur Physics, Koblenz, Germany, Tech. Rep. 16/1992, 1992.

[11] N. K. Treadgold and T. D. Gedeon, "Exploring constructive cascade networks," *IEEE Trans. Neural Networks*, vol. 10, pp. 1335–1350, 1999.

[12] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new adaptive merging snd growing algorithm for desiging artificial neural networks," *IEEE Transactions on System, Man and Cybernetics- Part B:Cybernetics*, vol. 39, no. 3, pp. 705–722, 2009.

[13] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "Covnet: A cooperative coevolutionary model for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 575–596, May 2003.

[14] S. M. Kamruzzaman and M. M. Islam, "An algorithm to extract rules from artificial neural networks for medical diagnosis problems," *International Journal of Information Technology*, vol. 12, no. 8, pp. 41–59, 2006.

[15] M. I. B. Shahid, M. M. Islam, M. A. H. Akhand, and K. Murase, "A new algorithm to design multiple hidden layered artificial neural networks," in *Proceedings of Joint 3nd International Conference on Soft Computing and Intelligent Systems and 7th International Symposium on Advanced Intelligent Systems (SCIS-ISIS 2006)*, September 2006, pp. 463–468.