Available online at www.elixirpublishers.com (Elixir International Journal)

Sustainable Architecture

Elixir Sustain. Arc. 65 (2013) 19736-19744

ISSN: 2229-712X

Architecture design and time complexity of artificial neural network using evolutionary algorithm

G.V.R. Sagar* and K. Anitha Sheela
G.P.R. Engg. College, Kurnool AP, 518007, India.
JNT University, Hyderabad AP, India.

ARTICLE INFO

Article history:
Received: 17 October 2013;
Received in revised form:
25 November 2013;
Accepted: 4 December 2013;

Keywords
Evolutionary algorithm,
Crossover,
Mutation,
Time complexity.

ABSTRACT

This paper highlights the role of new Evolutionary Algorithm (EA) in designing Artificial Neural Networks (ANNs). The proposed EA has been used for two purposes. One is generalization of architecture. In this, the optimal adaptive architecture is achieved by using evolutionary crossover and mutation. The adaptive strategy merges or adds hidden neurons based on the learning ability of hidden neurons or the training progress of ANNs. The mathematical frame work is also discussed. The other is the Time Complexity of ANN to reach the global minima using two selection processes. The proposed EA has been tested on a number of benchmark problems in machine learning and ANNs, including breast cancer, diabetes, heart problems and for time complexity N-Bit Parity is used. The experimental results show that proposed EA can design compact ANN architectures with good generalization ability, compared to other algorithms with good time complexity.

Introduction

ARTIFICIAL neural networks (ANNs) have been used widely in applications like system identification, signal processing, classification, and pattern recognition. Most applications were developed using feed-forward ANNs and the back-propagation (BP) learning algorithm [1] [19]. The important issue in using ANNs is to choose their architectures appropriately. Whereas a too large architecture may overfit the training data a too small architecture may underfit the training data. Both overfitting and underfitting cause bad generalizations of ANNs. So, it is necessary to design ANNs automatically and they can solve different problems efficiently. There are many algorithms available for designing ANNs automatically, such as constructive, pruning, constructive–pruning, and regularization algorithms [2]–[4]. A constructive algorithm [18] adds hidden layers, neurons, and connections to a minimal ANN architecture. A pruning algorithm deletes unnecessary hidden layers, neurons, and connections from an oversized ANN. A constructive–pruning algorithm is a hybrid approach. In addition, evolutionary approaches, such as genetic algorithms [5], evolutionary programming [6], [7], and evolution strategies [8], have been used extensively in designing ANNs automatically.

The main problem in designing ANNs using constructive, pruning, constructive–pruning, and regularization algorithms is that they use a predefined, fixed, and greedy strategy. Thus, these algorithms are susceptible to becoming trapped at *architectural local optima* [9]. The proposed work presents a new Evolutionary algorithm (EA) that uses one-point crossover and adaptive merging and adding process at the mutation level in designing ANNs. The decision of when to merge or add hidden neurons, is completely dependent on the improvement of hidden neurons' learning ability or the training progress of ANNs. A time complexity analysis is also made on the proposed EA.

Evolutionary Algorithms (EAs):

Evolutionary Algorithms (EAs) refer to a class of population based stochastic search algorithms developed [20] from ideas and principle of natural evolution. They are a class of stochastic optimization algorithms inspired by biological process that allows populations of neurons to adapt genetic inheritance and survival of the fittest. The architecture design is crucial in ANN because it has significant impact on network information processing capabilities. For a given learning task, an ANN with only a few connections and neurons may not be able to perform the task at all, due to its limited capability, while an ANN with a large number of nonlinear neurons and connections may overfit noise in the training data and fail to have good generalization ability.

Tele:
E-mail addresses: nusagar@gmail.com

The main advantage of this approach is that it can avoid the architectural local optima problem [9], [10]. However, the evolutionary approach is quite demanding in both time and user-defined parameters [2]. Moreover, it is necessary to find a set of optimal control parameters so that an evolutionary process can balance exploration and exploitation in finding good quality solutions. This can be overcome, by using proposed evolutionary learning process applied on feed-forward ANN architecture.

Mathematical Model of Evolutionary Algorithm:

The mathematical frame work in this section gives a general description of a proposed evolutionary search in EA. The schemata correspond to invariant subsets of the search space and the schema theorem can be reformulated in proposed framework. In the proposed evolutionary algorithm the architecture search space denoted by $\Omega = \Pi_{i=1}^{n} A_i$. P is the population of architectures which consisting of m individuals and n number of layers. So P is an m x n order given by equation (1.1). W is the weight population with same size of P.

$$P = \begin{bmatrix} a_{11} & a_{12} & . & . & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & . & . & a_{2n} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & . & . & a_{mn} \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & . & . & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & . & . & w_{2n} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w_{m1} & w_{m2} & . & . & . & . & w_{mn} \end{bmatrix} \qquad 1.1$$

Frame work: A population P of size m {1, 2,…., m} is considered and elementary steps of initial selection, partition and recombination are applied on P. The first elementary step is the initial selection is applied on P and new population P^1 is generated.

$$P = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \text{with } x_i \in \Omega \qquad 1.2$$

The fitness evaluation for individuals of P is

$$\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_m \end{bmatrix} \begin{matrix} \to f(x_1) \\ \to f(x_2) \\ . \\ . \\ . \\ \to f(x_{1m}) \end{matrix} \qquad 1.3$$

The ‘selection’ of individuals in P^1 is based on probability rule $\frac{f(x_j)}{\Sigma_{l=1}^{m} f(x_i)}$ where f is the fitness function given as f = 1/mean square error.

$$P^1 = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_m \end{pmatrix} \qquad 1.4$$

The individuals having small fitness values are not allowed into P^1 at all. This is to institute the natural survival of the fittest principle.

The second elementary step is recombination. This is schema based recombination, i.e., the entire search space Ω is divided into sub group or sub algorithms called schemata. The resultant population P^{l} from first elementary step is then partitioned into k number of disjoint tuples for mating. This is based on some probability distribution p_m. This is represented by set K = {P_1,P_2,……P_k}, where each partition P is a k-tuple of integers $q=(q_1, q_2, ...q_k)$. For instance, if the partition selected based on p_m is $K=\{(i_1^1, i_2^1, ...i_{q1}^1), (i_1^2, i_2^2, ...i_{q2}^2), ..., (i_1^j, i_2^j, ...i_{qj}^j) ....\}$ the corresponding tuples are

$$Q_1=\begin{pmatrix} y_{i_1^1} \\ y_{i_2^1} \\ \cdot \\ \cdot \\ y_{i_{q1}^1} \end{pmatrix} \quad Q_2=\begin{pmatrix} y_{i_1^2} \\ y_{i_2^2} \\ \cdot \\ \cdot \\ y_{i_{q2}^2} \end{pmatrix} \quad .... \quad Q_j=\begin{pmatrix} y_{i_1^j} \\ y_{i_2^j} \\ \cdot \\ \cdot \\ y_{i_{qj}^j} \end{pmatrix} \quad .... \qquad 1.5$$

One-point crossover Transform:

For one point crossover, let L_i and R_i are the crossover operations on two subspaces Q_j and Q_{j+1} with n layered architectures. The family of crossover transformations F is given as

$$F=\{L_i | 0 \le i \le n, L_i = L_{\{1,2,....i\}}\} \cup \{R_i | 0 \le i \le n, L_i = L_{\{1,2,....i\}}\} \qquad 1.3$$

For instance, if a = (a_1, a_2, …, a_n), b=(b_1, b_2, ….,b_n), $a \in Q_j$ and $b \in Q_{j+1}$ then, each element of a and b are represents the matrices of the layers in the two parent architectures. Each layer has m number of neurons with weight matrix W. The probability distribution of family F is

$$L_i(a,b)=(a_1, a_2, ...a_i, b_{i+1}, b_{i+2}, ...b_n) \qquad 1.4$$

$$R_i(a,b)=(b_1, b_2, ...b_n, a_{i+1}, b_{i+2}, ...a_n) \qquad 1.5$$

The above process is called as one-point crossover and the new off-spring population is P^{11}. On completion of a cycle, the procedure starts all over again, with the initial population p^m and the cycle is repeated a number of times depending on the problem.

$$P^{11}=\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_m \end{pmatrix} \qquad 1.6$$

Mutation:

Finally, mutation is the process with small probabilities that replace z_i with $F(z_i)$ according to the evolutionary process given in the next section, for chosen F$\in$ M. This creates a new population P^{111}.

$$P^{111}=\begin{pmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ \cdot \\ w_m \end{pmatrix} \qquad 1.7$$

Architecture Optimization:

Designing architecture is an important issue in the evolution of an ANN. The proposed evolutionary algorithm used the one point or cutting point crossover to improve the behavior between parents and off-springs. The cutting points are independently extracted for each parent according to equations (1.7) and (1.8). The cutting point is taken only between the hidden layers; this means that a new evolutionary weight matrix has been created to make connection between two layers at the cutting points in the parents producing two off-springs, so that the population is maintained constant.

The proposed EA used an adaptive search strategy in designing ANN. The adaptive strategy is evolutionary merging and adding of neurons based on the learning ability of hidden neurons or layers in ANN. Fig (1.0) shows the mutation flow chart.

Neuron Merging: Created off-springs carried from crossover consist of M number hidden neurons in the hidden layer. where M is selected by the user. All connection weights remain same. An epoch counter (μ_i) is initialized with zero, this count is used to count the number of epochs of hidden neurons and is trained. Mean Square Error (E) value is computed on the training set. If the termination criterion is satisfied, the training process is stopped and final network architecture gives the optimized ANN. Each hidden neuron is labeled with significance η_i. The significance of each hidden neuron h_i [11] is computed. Train the ANN on the training set for a number of epochs (τ). Increment the epoch count as follows for i = 1,2…..,N,

$$\mu_i = \mu_i + \tau \qquad 1.8$$

where N is the number of hidden neurons in the existing architecture. Initially N and M are the same.

The neuron merging in EA is based on significance η_i. It is computed using equation (1.9) for the hidden neuron h_i.

$$\eta_i = \frac{\sigma_i}{\sqrt[s]{\mu_i}} \quad \text{Where } \sigma_i \text{ is the standard deviation.} \qquad 1.9$$

Significance η_i is small when its standard deviation σ_i and/or its number of training epochs μ_i is large. The smaller the value of η_i, the less significant h_i is. A less significant hidden neuron delivers constant information to the neurons of output layer. The next process of merging is to compute the correlations between s-labeled hidden neuron and other neurons in ANN. The EA uses the Pearson product-moment correlation coefficient to measure the correlation and it is denoted by C_{ij}. It is the correlation between s-labeled hidden neuron i and the unlabeled hidden neuron j and is given as

$$C_{ij} = \frac{\sum_{p=1}^{P}(h_i(p)-\bar{h}_i)\,(h_j(p)-\bar{h}_j)}{\sigma_i\sigma_j} \qquad 2.0$$

Where $h_i(p)$ and $h_j(p)$ are the outputs of hidden neurons i and j for the example p in the training set, the variables $\bar{h}_i$ and $\bar{h}_j$ are the mean values and σ_i and σ_j are standard deviation of h_i and h_j, respectively.

The EA merges two correlated neurons, for instant h_a and h_b, produces a neuron h_m. The algorithm assigns the input and output connection weights of the h_m as follows

$$w_{mi} = w_{ai} + w_{bi} \qquad i = 1,2,\ldots..p \qquad 2.1$$

$$w_{jm} = w_{ja} + w_{jb} \qquad j = 1,2,\ldots..q$$

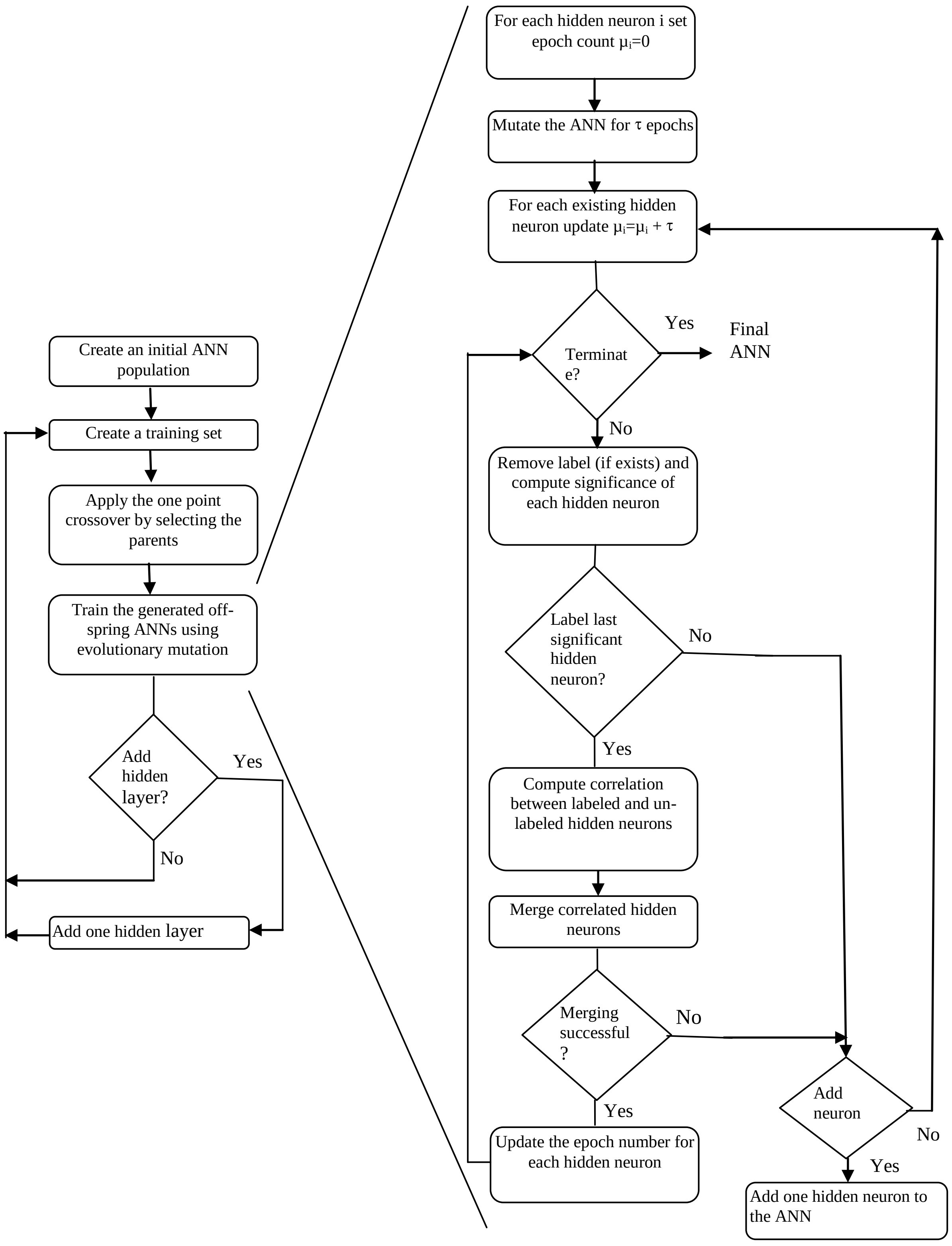
Where p and q are the number of neurons in the predecessor and successor layers of the ANN. The neuron merging is carried out based on, whose contribution is negligible with respect to the overall network output and the decision is based on the selection criterion given in the above equation 2.0.

Neuron Addition: The EA uses a simple criterion to add a hidden neuron in an ANN. This is based on the training error progress of the ANN. When the error of the ANN does not reduce by an amount ε after training epochs τ, EA assumes that, it is necessary to add hidden neurons in the ANN. Here ε and τ are two user specified parameters. The neuron addition criterion is given as

$$E(t) - E(t+\tau) \leq \varepsilon \qquad t = \tau, 2\tau, 3\tau \ldots\ldots. \qquad 2.2$$

Where $E(t)$ and $E(t+\tau)$ are the training errors at epochs t and $(t+\tau)$, respectively. EA adds a hidden neuron by splitting an existing hidden neuron of an ANN. Two neurons are created by splitting and have the same number of connections as off-spring neuron.

Proposed Evolutionary Algorithm Flow Chart:



Create an initial ANN population
Create a training set
Apply the one point crossover by selecting the parents
Train the generated off-spring ANNs using evolutionary mutation
Add hidden layer?
Yes
No
Add one hidden layer
For each hidden neuron i set epoch count μi=0
Mutate the ANN for τ epochs
For each existing hidden neuron update μi=μi + τ
Terminate?
Yes
Final ANN
No
Remove label (if exists) and compute significance of each hidden neuron
Label last significant hidden neuron?
No
Yes
Compute correlation between labeled and un-labeled hidden neurons
Merge correlated hidden neurons
Merging successful?
No
Yes
Update the epoch number for each hidden neuron
Add neuron
No
Yes
Add one hidden neuron to the ANN

Figure 1.0 Flow hart of proposed EA on ANN

Computational Time Complexity of Proposed Evolutionary:

A few optimization problems are very hard to solve using the EAs and may take very long time to reach the global optima. The take-over time in terms of generation is the mean hitting time (τ) of EA [12] [17]. EA may take very long time i.e., exponential number of generations to find the global optima is called wide-gap problems. The results are verified on well known benchmark problems like N-Bit Parity. The empirical results are almost same as the theoretical results. Here, first a truncation selection with mutation (selection I) is applied. Second, a tournament selection with mutation operator (selection II) is applied.

Benchmark Real Data Classification Problems:

The proposed method of optimization algorithm is applied to benchmark problems using the feed-forward architecture with a bias of +1 input for hidden layer and output layer. All the data, applied to the training and test sets, are acquired from the UCI Machine Learning Repository [13]. The number of training and test data sets is given in Table 1. Testing error rate (TER) is one more paraeter in real time data classification problems, which refers to the percentage of wrong classification produced by ANNs on the testing set.

i) Pima-India-Diabetes dataset problems.

ii) SPECT Heart data set

iii) Breast-Cancer dataset problem

Experimental Setup EANN: The evolutionary process attempts to crossover and mutate weights before performing any structural or topology crossover and mutation. The weights of ANNs were initialized to random values in the range between −0.5 and +0.5.For evolving connection weights population size in EA is taken as 20 and 10 independent trials have been conducted to get the generalized behavior. For architecture the number of training generations for partial training □ is set to 20, the values ε and η are set to 1E-06 and 0.06 respectively. The value of T used in the termination criterion is set to 3 (for real classification problems). Condition of terminating criteria is based on training error.

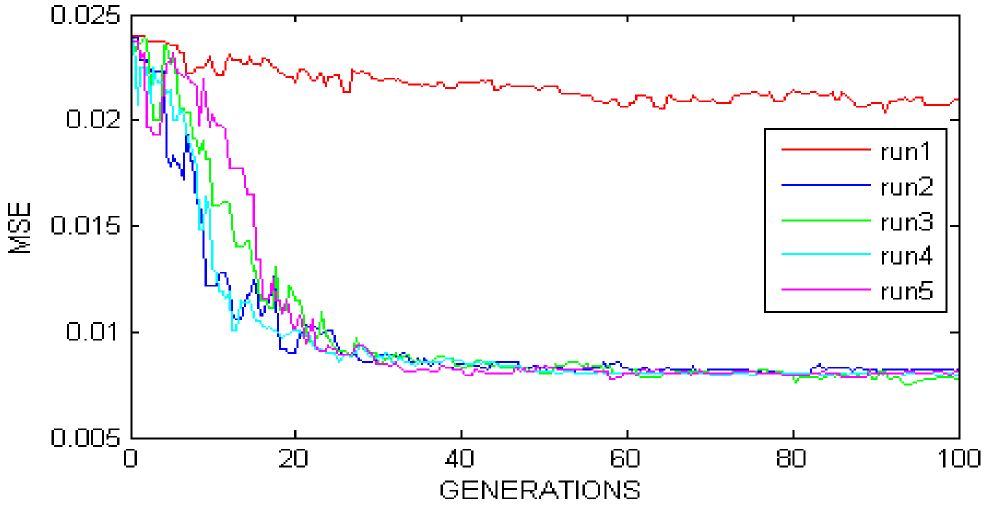


Figure 2.0 Performance of Evolutionary ANN for Pima Indians diabetes with initial size of [9 4 5 1 2].

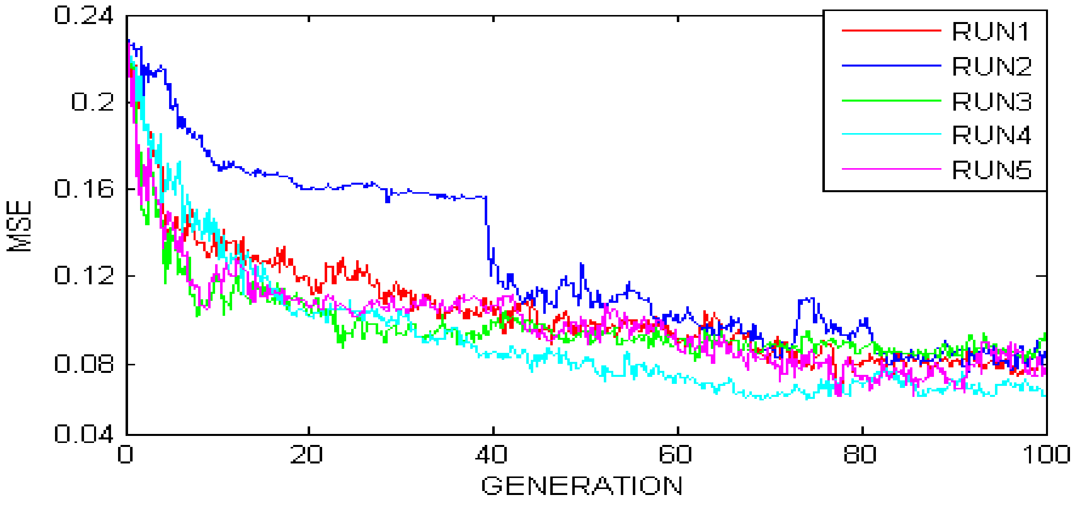


Figure 3.0 Performance of Evolutionary ANN for SPECT Heart dataset with initial size of [14 4 5 1 2].

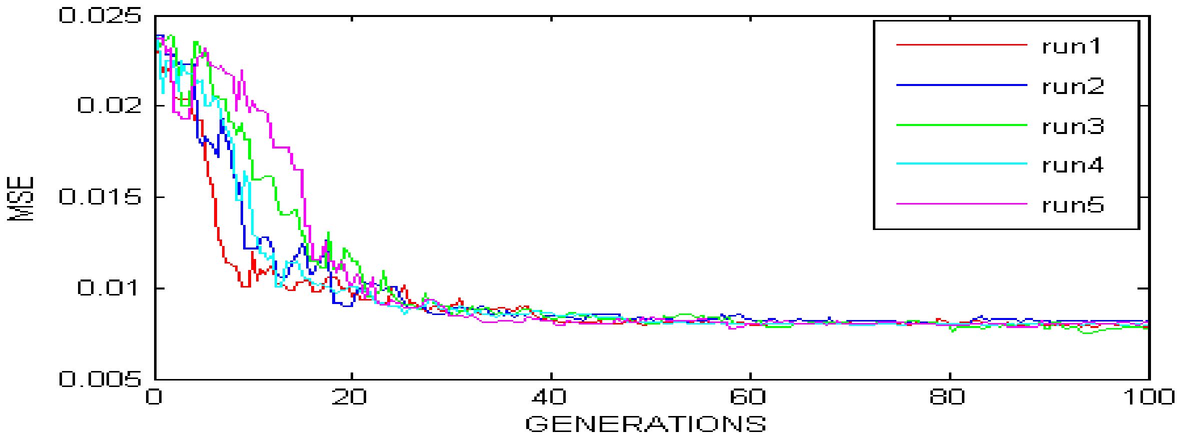


Figure 4.0 Performance of Evolutionary ANN for Breast Cancer dataset with initial size of [11 4 5 1 2].

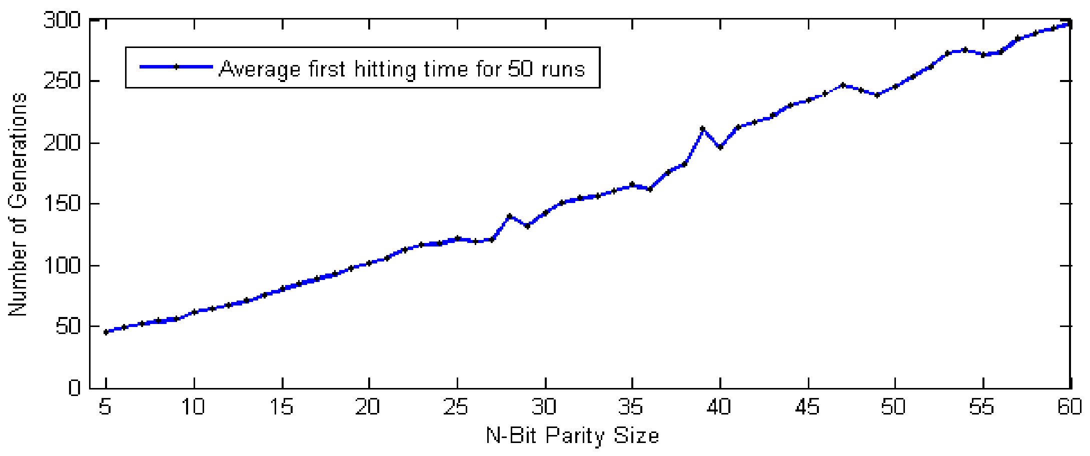


Figure 5.0. The Average first hitting time of EA with selection I.

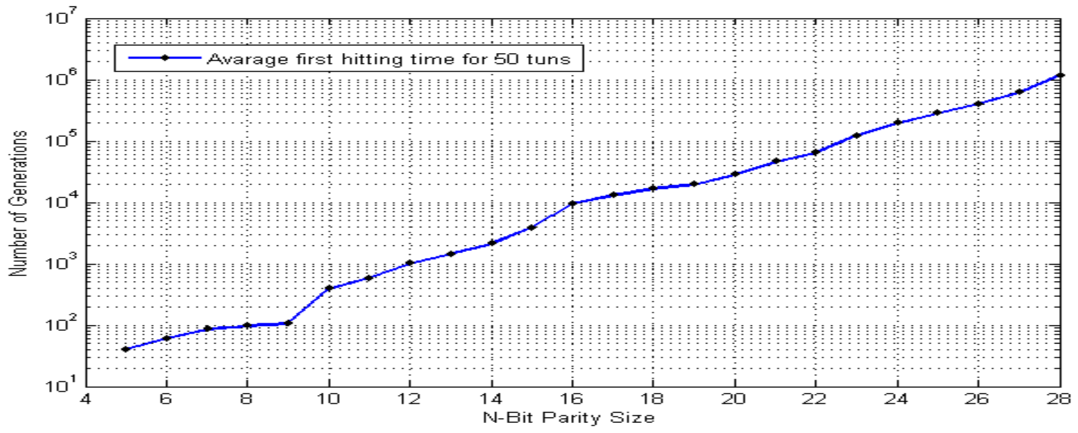


Figure 6.0. The Average first hitting time of EA with selection II

Result Analysis: The performance results on three benchmark real data classification problems are obtained. First by, for Pima Indians the average training error rate (TER) of 23.5 is obtained. The resultant architecture is optimized with an average of 3.02 hidden neurons and 1.2 average hidden layers. The minimum average mean square error is 8.6214E-3 as shown in Fig (2.0) and the average percentage of mse performance of the architecture is equal to 77.50. Second by, for SPECT Heart problem the average TER is equal to 13.5 over 10 trial runs. The resulting architecture is tested using 67 test sets and the minimum average mean square error obtained is 7.7264E-3 as shown in Fig (3.0) and minimum average number of hidden neurons is 3.41 with average number of hidden layers of 1.2. The average percentage of performance of proposed EA on the architecture is equal to 85.84. Third by, for the Breast Cancer problem, the optimized network with average hidden neurons and layers are 2.01 and 1.1 is obtained. The best average TER of 3.0 is achieved. The optimized architecture is tested with 240 test set vectors so that the mean square error is 5.3614E-3 is obtained and is as shown in Fig (4.0). The average percentage of performance of proposed EA on the architecture is equal to 98.5. The proposed EA also compared on number of hidden neurons, generations and TER with some different algorithms like BCA, BTA and BCPA [14], [[15], [16] given in Table 2.

The time complexity results are shown in Fig (5.0) for truncation selection. This result is averaged over 50 independent trials with smaller population size and Fig (6.0) shows the results of tournament selection and the number of generations which is averaged over 50 independent trials with possible larger population. On observation of time complexity results, the selection I pressure stores the best individuals with lower population size and to finds the path quickly but it fails for higher population. The results of selection II pressure stores the best or worst individual as seed individuals and the resultant generations are exponential. This selection pressure solves the problem with sufficiently higher selection pressure and larger population. So, for beginning it is best to use the lower selection pressure like selection I and after some epochs, to reach along the path higher selection pressure selection II is used.

Conclusion:

The proposed EA performed well with respect to the mean square error, adaptive optimal architecture and generalization when compared to the different algorithms in the literature. The time complexity shows that the initial training takes the selection I and the after some epochs to reach along the path, higher selection pressure like selection II is quit effective.

One of the future improvements to proposed EA would be to reduce the number of parameters or make them adaptive. In addition, the use of a different significance criterion in the merging operation of EA would also be an interesting future research topic. Since proposed EA has been applied to the classification problems, it would be interesting to study how well EA would perform on regression problems.

Table 1. Performance of EA on Three Real Time dataset problem.

Parameter	Experimental Results		
	Pima-Indians	SPECT-Heart	Cancer
Number Of Runs	10	10	10
Average Number Of Generations	61	103	52
Number of Training patterns used	500	200	400
Average Training Set Accuracy in percentage	76.5	87.2	97.0
Number of Test patterns used	268	67	240
Average Test Set Accuracy in percentage	77.5	85.12	98.02
Initial Number of Hidden layers / Neurons	2 / [5 4]	2 / [5 4]	2 / [5 4]
Final Number of Hidden layers / Neurons (Resulted NN)	1.2 / [3.02]	1.4 / [3.41]	1.1/ [2.01]
Population size	50	50	50
Number of inputs	09	14	11
Number of outputs	01	01	01

Table 2. Comparison of different methods based on architecture size and TER for three real dataset classification problems.

Algorithm or Method	Diabetes problem			SPECT-Heart problem			Cancer problem		
	Number of		TER	Number of		TER	Number of		TER
	Hidden neurons	Genera-tions		Hidden neurons	Genera-tions		Hidden neurons	Genera-tions	
BCA	5.96	467.5	26.04	3.42	173.4	20.34	2.20	290.2	1.92
BPA	5.56	409.6	26.25	3.12	161.4	19.93	1.60	263.3	1.89
BCPA	5.80	501.3	26.22	3.26	190.7	20.43	2.12	311.5	1.95
CNNDA	4.2	235	--	19.8	591.5	--	3.5	415.5	--
NCA	--	--	--	2.62	170	18.1	2.08	293	0.97
Proposed EA	3.02	358.7	23.5	3.41	190.6	15.1	2.01	228.2	2.86

References:

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing*, vol. I, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge,MA: MIT Press, 1986, pp. 318–362.

[2] T. Y. Kwok and D. Y. Yeung, “Constructive algorithms for structure learning in feed forward neural networks for regression problems,” *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 630–645, May 1997.

[3] R. Reed, “Pruning algorithms—A survey,” *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.

[4] F. Girosi, M. Jones, and T. Poggio, “Regularization theory and neural networks Architectures,” *Neural Comput.*, vol. 7, no. 2, pp. 219–269, Mar. 1995.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[6] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[7] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE Press, 1995.

[8] H.-P. Schwefel, *Numerical Optimization of Computer Models*. Chichester, U.K.: Wiley, 1981.

[9] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.

[10] X. Yao, “Evolving artificial neural networks,” *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[11] Md.Monirul et al., “A new adaptive merging and growing algorithm for designing artificial Neural,” *IEEE Trans. Man, and Cybernetics.*, vol. 39, no. 3, Apr. 2009, pp. 705–722.

[12] Chen et al, “New Approach for analyzing average time complexity of EAs on Unimodal problems”, IEEE Transactions on systems, Man, and Cybernetics – Part B, Cybernetics, Vol.39, No.5, October – 2009.

[13] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases,1998.

[14] R. Reed, “Pruning algorithms—A survey, ”*IEEE Trans. Neural Netw.*, vol.4,no.5,pp.740–747,Sep.1993.

[15] Y. LeCun, J.S.Denker, and S.A.Solla, “Optimal brain damage,” in *Proc. Advances Neural Inform. Process. Syst.*, D.S. Touretzky, Ed.SanMateo, CA.Morgan Kaufmann,1990,vol.2, pp.598–605.

[16] B.Hassibi and D.G.Stork, “Second-order derivatives for network prun ing:Optimal brain surgeon,” in *Proc. Advances Neural Inform. Process. Syst.*, C.Lee, S.Hanson, andJ.Cowan, Eds.San Mateo, CA:Morgan Kaufmann,1993,vol.5,pp.164–171.

[17] He, X. Yao, Drift analysis and average time complexity of evolutionary algorithms, Artif. Intell. 127 (1) (2001) 57_85.

[18] H. C. Andersen and A. C. Tsoi, “A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm,” *Complex Syst.*, vol. 7, no. 4, pp. 249–268, 1993.

[19] Y. Hirose, K. Yamashita, and S. Hijiya, “Back-propagation algorithm which varies the number of hidden units,” *Neural Networks*, vol. 4, no. 1, pp. 61– 66, 1991.

[20] M. W. Hwang, J. Y. Choi, and J. Park, “Evolutionary projection neural networks,” in *Proc. 1997 IEEE Int. Conf. Evolutionary Computation, ICEC’97*, pp. 667–671.