DESIGN OF AN INTEGRATED NEURO-GENETIC PROCESSOR FOR PATTERN RECOGNITION APPLICATIONS

A dissertation submitted in partial fulfillment of the requirement for the award of degree of

Master of Technology
In
VLSI Design and CAD

Submitted by:

SUNIL KUMAR

Roll No: 601161014

Under the guidance of:

Dr. Ravi Kumar

Assistant Professor, ECED

Thapar University, Patiala

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY, Patiala

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147 004 (PUNJAB)

DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, **"Design of an Integrated Neuro-Genetic Processor For Pattern Recognition Applications"** in partial fulfillment of the requirement for the award of degree of **Master of Technology** in **VLSI Design & CAD** submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. RAVI KUMAR, Assistant Professor, ECED** and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University/Institute for the award of degree.

Date: 25/6/13

Sunil Kumar
Roll No. 601161014

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 25/06/2013

Dr. Ravi Kumar
Assistant Professor, ECED
Thapar University, Patiala

Countersigned by:

Head
ECED, Thapar University
Patiala-147004

Dean of Academic Affairs
Thapar University
Patiala- 147004

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to **Dr. RAVI KUMAR, Assistant Professor**, Electronics & Communication Engineering Department, Thapar University, Patiala for his guidance and support throughout this thesis work. I am really very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am thankful to the **Head of Department, Professor (Dr.) RAJESH KHANNA** and **PG Coordinator, Dr. KULBIR SINGH (Associate Professor)** of Electronics & Communication Engineering Department for their encouragement and inspiration for the execution of this thesis work.

I am also thankful to the entire faculty and staff of Electronics & Communication Engineering Department for the help and moral support which went along the way for the successful completion of this thesis work.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

Sunil Kumar
601161014

ABSTRACT

One of the key concerns of modern VLSI design is realization of massively parallel architecture for high throughput and fast data processing. Artificial neural network are best example of such architecture whose implementation suffers from various bottlenecks including slow training and massive consumption of computational resources.

This thesis is an effort to overcome the limitation of backpropagation trained artificial neural network by adopting a hybrid Neuro-Genetic processing and introducing novel genetic operators. A new mutation operator derived from statistical analysis has been proposed and pruning of redundant weights has been accomplished.

Furthermore, an improved mutate-discard-crossover scheme has been implemented which retains the fittest weights. Simulation experiments confirm the efficacy of the proposed techniques. With the same amount of error being obtained in less number of epochs and lesser computational burden.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION TO PATTERN RECOGNITION

Pattern recognition as the name signifies, is the study of classification of object into different categories and classes. In other words, it is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns. It is a fundamental component of artificial intelligence.

The classification scheme is usually based on the availability of a set of patterns that have already been classified or described. This set of patterns is termed the training set, and the resulting learning strategy is characterized as supervised learning. Learning can also be unsupervised, in the sense that the system is not given any a priori labeling of patterns, instead it itself establishes the classes based on the statistical regularities of the patterns.

A wide range of algorithms exist for pattern recognition, from naive Bayes classifiers and neural networks to the powerful SVM decision rule.

Neural networks (NNs) are simplified models of the biological nervous systems. An NN can be said to be a data processing system, consisting of a large number of simple, highly interconnected processing elements(artificial neurons), in an architecture inspired by the structure of the cerebral cortex of the brain. The interconnected neural computing elements have the quality to learn and thereby acquire knowledge and make it available for use.

This thesis investigates the application of an Artificial Neural network with use of efficient optimization method, known as Genetic algorithm, to the field of pattern recognition. In this project, we devise a methodology for identifying the species of an iris amongst three classes based on four distinctive features. For this a standard IRIS dataset is taken from the UCI machine learning repository website (http://archive.ics.uci.edu/ml/datasets/Iris) for the input to the neural networks. The Iris dataset contains three classes of 50 instances each, where each class refers to a type of iris plant virginica, setosa and versicolor. Every instance consists of 4 dimensions- sepal length, sepal width, petal length and petal width.

The data is normalized to obtain a matrix with values ranging from 0 to 1. After analyzing the data it was found the one class was linearly separable from the other two but the other two classed could not be separated linearly as shown in Fig. 1.1

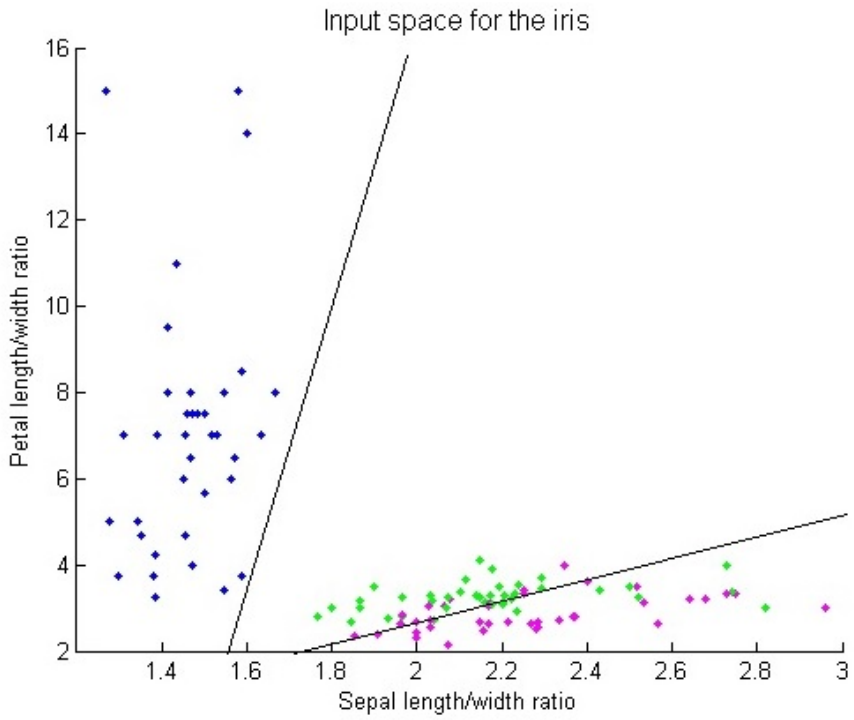

Fig. 1.1: Scatter Plot for Iris data sample

1.1 Need of Pattern Recognition

Today, Pattern Recognition and Machine Learning techniques are employed in many applications like *speech recognition, text pattern classification, identification of human faces, fingerprints or iris patterns and optical character recognition (OCR)* for the automatic digitization of scanned documents. Because of their great potentials and wide usage in many areas, Pattern Recognition and Machine Learning are also subject of large interest in academics and research. Nonetheless, applying these techniques usually both requires professional expertise in the field of Machine Learning, as well as software engineering skills in order to integrate into real world applications. Furthermore, the development and adaptation of Pattern Recognition and Machine Learning systems most often differ to those of traditional software systems, since the methods used in such systems are data-driven, i.e. their performance strongly depends on the problems and domains they are applied to.

1.2 Learning Techniques in Artificial Neural Networks

ANNs constitute a large family of learning rules which are derived analytically and statistically with an aim to find an optimum weight vector for the given function approximation task. All the learning paradigms can be classified into some rough categories as follows:

Fig. 1.2 depicts the hierarchical representation of the types of learning algorithms used for ANNs. These learning mechanisms are explained below.

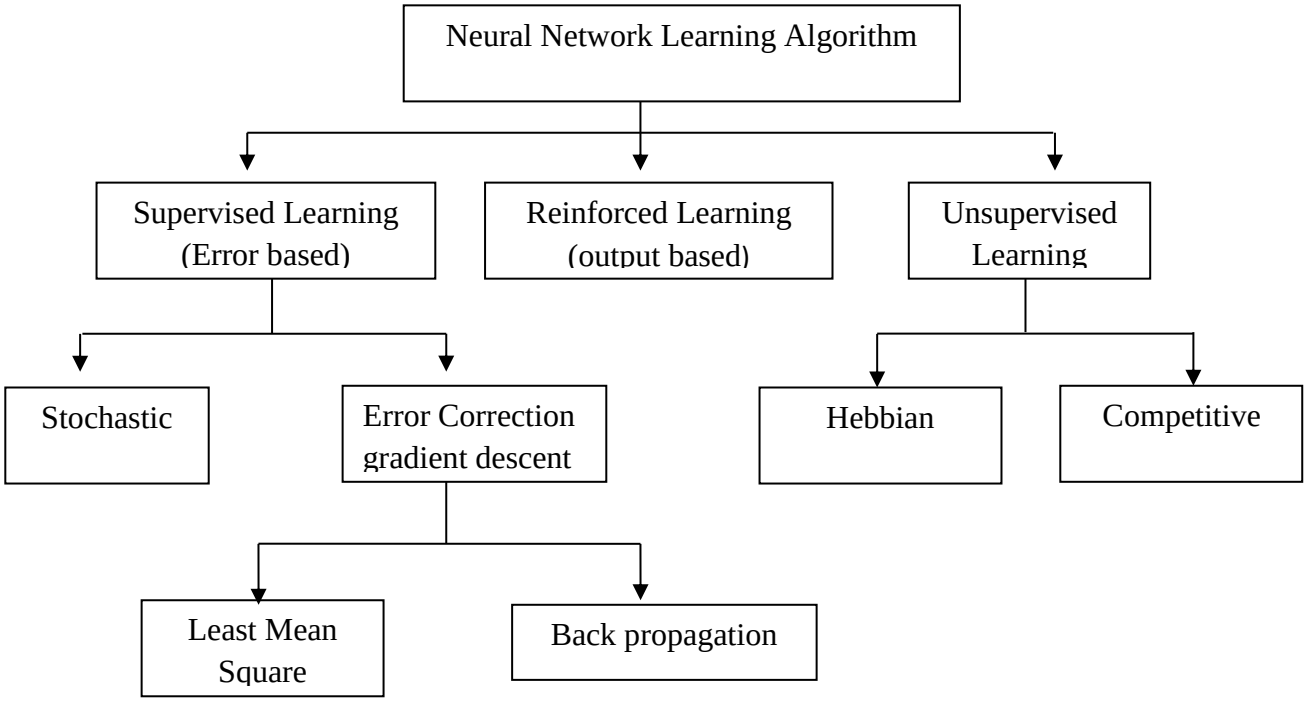

Fig. 1.2: Classification of Learning Algorithms for ANNs

1.2.1 Unsupervised Learning

In this learning method, the target output is not presented to the network. It is as if there is no teacher to present the desired patterns and hence, the system learns of its own by discovering and adapting to structural features in the input patterns. A suitable distance measure is chosen for taking decisions for assigning a particular sample to a class which is not predefined. Most of the clustering algorithms fall in this category where the even the number of clusters present in the data is not known.

1.2.2 Supervised Learning

In this method, every input pattern that is used to train the network is associated with an output pattern, which is the target or the desired pattern. A teacher is assumed to be present during the learning process, when a comparison is made between the network's computed output and the correct expected output, to determine the error, the error can then use to change network parameters, which result in an improvement in performance.

1.2.3 Reinforcement Learning

In this method, a teacher though available, doesn't present the expected answer or desired output but only indicates if the computed output is correct or incorrect. The information provided helps the network in its learning process. A reward is given for a correct answer computed and a penalty for a wrong answer. This method is not one of popular forms of learning.

Supervised and Unsupervised Learning are the most popular methods. Some of the widely used learning rules are now being presented below:

1.2.4 Hebbian Learning

This rule was proposed by Hebb (1949) and is based on correlative weight adjustment. This is the oldest learning mechanism by biology. In this, the input-output pattern pairs (X_i, Y_i) are associated by the weight matrix W, known as the correlation matrix. It is computed as

$$W = \sum_{i=1}^{n} X_i Y_i^T \tag{1.1}$$

However, there are some limitations of Hebbian learning e.g. it can learn only with bipolar inputs and targets.

1.2.5 Gradient Decent Learning

This is based on the minimization of error E defined in terms of the weights and the activation function of the network. Also, it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E. thus, if Δw_{ij} is the weight update of the link connecting the i^{th} and j^{th} neuron of the two neighboring layers, then Δw_{ij} is defined as

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} \tag{1.2}$$

where η is the learning rate parameter and $\partial E / \partial W_{ij}$is the error gradient.

1.2.6 Competitive Learning

In this method, those neurons which respond strongly to input stimuli have their weight updated. When an input pattern is presented, all neurons in the layer compete and the winning neurons undergo weight adjustment. Hence, it is a "winner-takes-all" strategy. Competitive learning is used as a first step to self-organization.

1.3 Data Classification Using Feed forward Networks

For the classification of real world data using feed forward ANNs, the most popular algorithm has been Backpropagation Algorithm owing to its simple structure and ease of training. The next section describes in brief fundamental working principle of Backpropagation algorithm.

1.3.1 Backpropagation Algorithm [1]

Backpropagation is a method of training multilayer artificial neural networks which uses the procedure of supervised learning. Supervised algorithms are error-based learning algorithms which utilize an external reference signal (teacher) and generate an error signal by comparing the reference with the obtained output. Based on error signal, neural network modifies its synaptic connection weights to improve the system performance. In this scheme, it is always assumed that the desired answer is known a priori".

Let the error signal at the output of neuron j at iteration n be defined by

$$e_j(n) = d_j(n) - y_j(n) \tag{1.3}$$

Now, we define the instantaneous value of the error energy for neuron j as $\frac{1}{2}e_j^2(n)$. Correspondingly, the instantaneous value $\xi(n)$ of the total error is obtained by summing $\frac{1}{2}e_j^2(n)$ overall neurons in the output layer;

We may thus write

$$\xi(n) = \frac{1}{2}\sum_{j=C}^{N} e_j^2(n) \tag{1.4}$$

where the set C include all the neurons in the output layer of the network.

Let N denote the total number of pattern contained in the training set. The *average squared error energy* is obtained by summing $\xi(n)$over all n and then normalizing with respect to the set size N, as shown by

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n) \qquad (1.5)$$

The instantaneous error energy $\xi(n)$, and therefore the average error energy ξ_{av}, is a function of all the synaptic weights and bias levels of the network which are also known as free parameters of the network. For a given training set, a cost function ξ_{av} is defined as a measure of learning performance. The objective of the learning process is to adjust the free parameters of the network to minimize ξ_{av}. Let us consider a simple method of training in which the weights are updated on a *pattern-by-pattern* basis until one epoch, that is, one complete presentation of the entire training set has been dealt with. The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the network. The arithmetic average of these individual weight changes over the training set is therefore an estimate of the true change that would result from modifying the weight based on minimizing the cost function ξ_{av}over the entire training set.

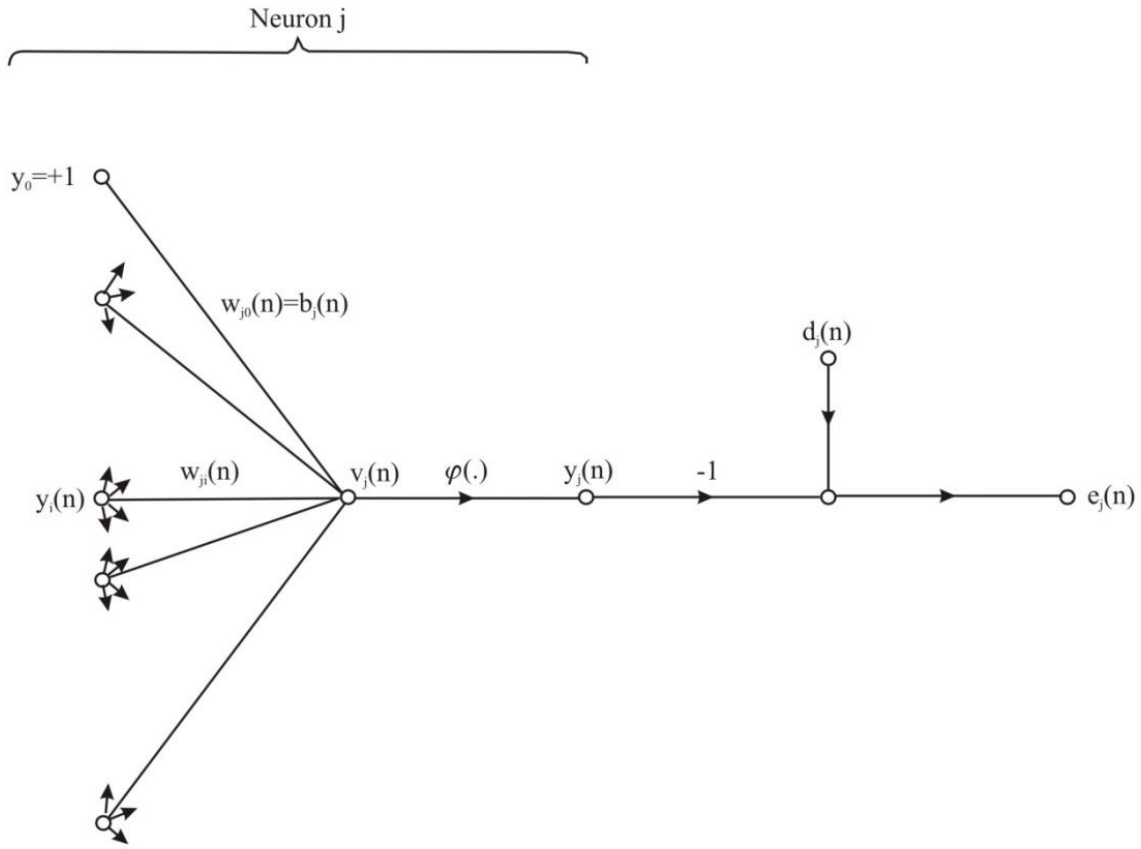

Fig. 1.3 Signal-flow graph highlighting the detail of output neuron j

Consider Fig. 1.3 which depicts neuron j being fed by a set of function signals produced by a layer of neuron to its left. The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron *j* is therefore

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n) \tag{1.6}$$

where m is total no. of inputs (excluding the bias) applied to neuron *j*. The synaptic weight w_{j0} (Corresponding to fixed input $y_0 = +1$) equals the bias b_j applied to neuron *j*. Hence the function signal appearing at the output of neuron *j* at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \tag{1.7}$$

The back propagation algorithm now applies a correction $\Delta w_{ji}(n)$ to a synaptic weight $w_{ji}(n)$, which is proportional to the partial derivative $\partial\xi(n)/\partial w_{ji}(n)$. According to the *chain rule* of calculus, the gradient can be expressed as:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{1.8}$$

The partial derivative $\partial\xi(n)/\partial w_{ji}(n)$ represents a *sensitivity factor*, determining the direction of search in weight space for the synaptic weight w_{ji}.

Differentiating both sides of Eq. (1.4) with respect to $e_j(n)$, we get

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n) \tag{1.9}$$

Differentiating both sides of Eq. (1.3) with respect to $y_j(n)$, we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{1.10}$$

Next, differentializing eq. (1.7) with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n)) \tag{1.11}$$

where the use of prime (on the right hand side) signifies differentiation with respect to the argument. Finally, differentiation Eq. (1.6) with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n) \tag{1.12}$$

The use of Eqn. (1.9) to (1.12) in (1.8) yields

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))y_i(n) \tag{1.13}$$

The correction of $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the *delta rule*

$$\Delta w_{ji}(n) = -\eta \delta_j(n) y_i(n) \tag{1.14}$$

where η is the *learning-rate* parameter of the back propagation algorithm. The use of minus sign in Eq. (1.14) accounts for *gradient descent* in weight space (i.e. seeking a direction for weight change that reduces the value of $\xi(n)$). Accordingly, the use of Eq. (1.13) in (1.14) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{1.15}$$

where the local gradient $\delta_j(n)$ is defined by

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \xi(n)}{\partial v_j(n)} \\ &= -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n)\varphi'_j(v_j(n)) \end{aligned} \tag{1.16}$$

The local gradient points to required changes in synaptic weights. According to eq. (1.16) the local gradient $\delta_j(n)$ for output neuron *j* is equal to the product of corresponding error signal $e_j(n)$ for that neuron and the derivative $\varphi'_j(v_j(n))$ of the associated activation function.

From Eq. (1.15) and (1.16) we note that a key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron*j*. In this context two distinct cases can be identified, depending on where in the network, neuron *j* is located. In case 1, neuron *j* is an output node. For an output neuron, a desired response is defined. However, for a hidden layer neuron no such desired response. This situation gives rise to what we call "credit assignment problem". When there is no defined output, how are we going to assess the error for that particular neuron? To solve this particular problem, we take two cases and apply the backpropagation algorithm.

Case 1: Neuron *j* is an output Node

When neuron is located in the output layer of the network, it is supplied with a desired response of its own. We may use Eq. (1.3) to compute the error signal $e_j(n)$ associated with this neuron; see Fig. 1.3. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$ using Eq. (1.16).

Case 2: Neuron *j* is a hidden Node

When neuron is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-propagation algorithm gets complicated. Consider the situation depicted in Fig. 1.4, which depicts neuron *j* as a hidden node of the network. According to Eq. 1.16, we may redefine the local gradient for hidden neuron *j* as

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \qquad (1.17)$$

$$= -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi'_j\left(v_j(n)\right), \text{ neuron } j \text{ is hidden}$$

where in the second line we have used Eq. (1.11).

To calculate the partial derivative $\partial \xi(n)/\partial y_j(n)$, we may proceed as follow. Form 4.4 we see that

$$\xi(n) = \frac{1}{2}\sum_{k \in C} e_j^2(n), \text{ neuron } k \text{ is hidden} \qquad (1.18)$$

which is Eq. (1.4) with index k used in place of index *j*. We have done so in order to avoid confusion with the use of index *j* that refers to a hidden neuron under case 2. Differentiating Eq. (1.18) with respect to the function signal $y_j(n)$, we get

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_i(n)} \qquad (1.19)$$

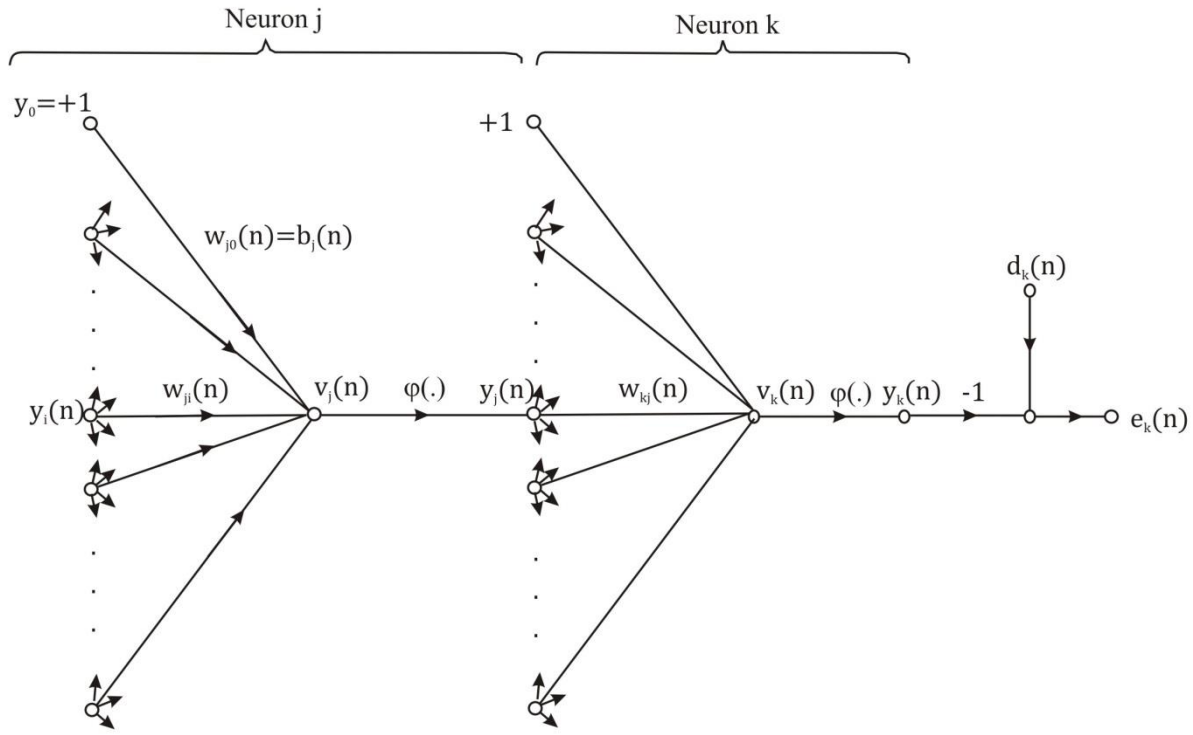

Fig. 1.4 Signal-flow graph highlighting the detail of output neuron *k* connected to hidden neuron *j*

Next we use the chain rule for the partial derivative $\partial e_k(n)/\partial y_j(n)$ and rewrite Eq. (1.19) in the equivalent form

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{1.20}$$

However, from Fig. 1.4, we note that

$$e_k(n) = d_k(n) - y_k(n) \tag{1.21}$$

$$= d_k(n) - \varphi_k(v_k(n)), \text{ neuron } k \text{ is an output node}$$

Hence

$$\frac{\partial e_k(n)}{\partial v_k(n)} = \varphi'_k(v_k(n)) \tag{1.22}$$

We also note from Fig. 1.4 that for neuron k the induced local field is

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n)\, y_j(n) \tag{1.23}$$

where *m* is the total number of inputs (excluding the bias) applied to neuron k. Here again, the synaptic weight $w_{k0}(n)$ is equal to the bias $b_k(n)$ applied to neuron k, and the corresponding input is fixed at the value +1. Differentiating Eq. (1.23) with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{1.24}$$

By using Eqs. (1.22) and (1.24) in (1.20) we get the desired partial derivative:

$$\begin{aligned} \frac{\partial \xi(n)}{\partial y_j(n)} &= -\sum_k e_k(n)\, \varphi'_j\left(v_j(n)\right) w_{kj}(n) \\ &= -\sum_k \delta_k(n)\, w_{kj}(n) \end{aligned} \tag{1.25}$$

where in the second line we have used the definition of the local gradient given in Eq. (1.16) with the index *k* substituted for *j*.

Finally, using Eq. (1.25) in (1.26), we get the *back-propagation formula* for the local gradient $\delta_j(n)$ as described:

$$\delta_j(n) = \varphi'_j\left(v_j(n)\right) \sum_k \delta_k(n)\, w_{kj}(n), \text{ neuron } j \text{ is hidden} \tag{1.26}$$

The factor $\varphi'_j(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in Eq. (1.26) depends solely on the activation function associated with hidden neuron j. The remaining factor involved in this computation, namely the summation over k, depends on two set of terms. The first set of terms, the $\delta_k(n)$ requires knowledge of the error signal $e_k(n)$, for all neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron *j*: see Fig. 1.4. The second set of terms, the $w_{kj}(n)$ consists of the synaptic weights associated with these connections.

The relations derived for the back propagation algorithm are now being summarized. First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron j defined by the delta rule:

$$\begin{pmatrix} Weight \\ correction \\ \Delta\omega_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning- \\ rate-parameter \\ \eta \end{pmatrix} \bullet \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} \bullet \begin{pmatrix} input.signal \\ of.neuron \cdot j \\ y_j(n) \end{pmatrix} \tag{1.27}$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, equals the product of the derivative $\varphi'_j(v_j(n))$ and the error signal $e_j(n)$ both of which are associated with neuron j; see Eq. (1.16).
2. If neuron j is a hidden node $\delta_j(n)$equals the product of the associated derivative $\varphi'_j(v_j(n))$ and the weighted sum of the δs computed for the neurons in the next hidden or output layer that are connected to neuron see Eq. (1.26).

It is evident that weight updation at any iteration involves computation of local gradient both for the output and hidden layer neurons. This puts a considerable computational burden on the simulator and hence poses a challenge to hardware implementation. In addition to it, there are a few more design challenges related to implementation of a BP-classifier which are being described in the next section.

1.4 Design Challenges

The working of back propagation algorithm is governed by equation 1.3 to 1.27 described in the previous section. It can be noted very easily that computation of local gradient for a hidden layer neuron is challenging task in itself especially when the neuron is in the hidden layer its output is not defined and hence credit assignment problem arises. Despite their simple architecture, requirement of back-propagating the error signal to compute local gradient sometimes becomes a limitation for the BP-network. For sparse and already clustered data, the BP algorithm performs exceptionally well. However, when the class separability of data is low then BP algorithm tends to getting stuck in local minima thus yielding suboptimal solution. In such a scenario, there are several challenges associated with designing an optimal classifier which are being enlisted as follows:

1. **Selecting an optimal architecture of ANN**: One of the greatest drawbacks of BP training is the experimental determination of the number of hidden layer neurons which requires many iterations of experimental training before the network trains well enough to receive the test data. It also modifies the statistical properties of the initialized weight vector, thus making the whole learning process totally undeterministic.
2. **Black-Box nature of BP-ANN**: The ANN behaves as a black-box during training with BP algorithm. It is thus a challenge to give the training process a firm analytical background.
3. **Unalterable weight vectors** : To classify the real world data, it is required that the weight vectors be modifiable during successive iterations more in a statistical sense than what is defined currently according to the back propagated error signals.
4. **Heavy consumption of computational resources:** Most of the computational time during BP training is lost in the computation of local gradient especially when training has proceeded for a considerable number of epochs thus we require to put in place such a system in which computation of local gradient could be avoided for certain number of epochs.

Keeping in view above mentioned design challenges, this thesis reports a novel endeavor in the form of Neuro-Genetic classifier which, to a large extent, has been able to do away with calculation of local gradient. Instead, it gives an optimized set of weights with which the ANN can be initialized and tested using standard data for classification purpose. Some aspects of this thesis are now being discussed in the next section.

1.5 Novel Aspects of This Thesis

1. This thesis presents for the first time a mutate-discard crossover scheme where mutation and crossover operations have been performed alternatively in tandem.
2. A novel mutation operator has been proposed which seeks to eliminate non-performing weights converging on a neuron. This scheme also defines analytically the fitness of weights in a statistical sense. Till now the authors reporting Neuro-genetic processing have defined the fitness of weights according to some fitness function commonly used in GA flow.
3. This thesis report results of exhaustive simulation work by performing experimentation with different ANN parameters (learning rate, momentum constant etc.) and different architectures.

CHAPTER 2

LITERATURE SURVEY

The paper presented by Fukumi et al. [4] proposed a method using a Genetic Algorithm (GA) with a partial fitness (PF) and a deterministic mutation (DM) to design a neural pattern recognition system for a rotated coin recognition problem. In this method, chromosomes in the CA were divided into several parts and their PFs were evaluated for GA operations. Furthermore, in this paper, DM based neural network learning was also introduced. It was shown that the proposed method was better than conventional GAs on convergence in learning and small-sized neural network were formed.

In a typical application of Neuro-Genetic processing, Srivastava et al. [5] had applied GA optimized weights to a BP trained ANN. For the high dimensionality data, back-propagation had great difficulty in training the neural classifier even with repeated restarts, and different weights initializations. To alleviate this problem, this paper reported a Neuro-Genetic processor in which a genetic algorithm with special MRX operator was implemented for processing electronic nose data corrupted with additive Gaussian noise.

In Hunter and Chiu [6], the author discussed the design of neural network and fuzzy logic controller using genetic algorithm, for real-time control of flows in sewerage network. The genetic algorithm here designs controllers and sets points by repeated application of a simulator. A comparison between neural network, fuzzy logic and bench mark controller performance was presented. Neural network and fuzzy logic controllers were found to have comparable performance although neural network can be successfully optimized for consistently.

Libelli and Alba [7] reported that making mutation a function of fitness produces a more efficient search. This function is such that the least significant bits are more likely to be mutated in high-fitness chromosome, thus improving their accuracy, whereas low-fitness chromosome have an increased probability of mutation, enhancing their role in the search. In this way, the chance of disrupting a high-fitness chromosome is decreased and the exploratory role of low-fitness chromosome is best exploited. The implications of this new mutation scheme had been assessed with the aid of numerical examples.

Bull [8] presents a simple model of genetic algorithm in such systems, with the aim of examining the effects of different types of interdependence between individuals. Using the model it was shown that, for a fixed amount of interdependence between coevolving individuals, the existence of partner gene variance and the level at which fitness is applied can have significant effects, as does the evaluation partnering strategy used.

At the hardware level, Imai et al. [9] had described the architecture of a scalable and high-speed GA processor, which was characterized by hardware-oriented approach based on distributed GA, optimized hierarchic pipelines for high-speed evolutions and flexible genetic operations corresponding to a given problem. Furthermore, this paper also described VLSI implementation of a processor-element to verify feasibility of the proposed architecture for applications. This work claimed to solve the problem of premature convergence and improper adjustment of GA parameters.

Kalra and Prakash [10] present the inverse kinematics solution of a robotic manipulator which required the solution of non-linear equations having transcendental functions and involving time consuming calculations. In this work, a Neuro-Genetic algorithm approach was used to obtain the inverse kinematics solution of a robotic manipulator. A multi-layered feed-forward neural network architecture was used whose weights were obtained during the training phase using a real-coded genetic algorithm. This training algorithm did not suffer from the usual drawbacks of the back-propagation learning algorithm.

The paper by Louis [11] investigated the effect of injection percentage on the performance of a case-injected genetic algorithm for combinational logic design. This particular algorithm is augmented with a case-based memory of past problem solving attempts which learns to improve performance on sets of similar design problems. This work proposed a scheme for periodic generation of the population so that a generalized solution could be injected into the GA flow.

The paper presented by Ramasubramanian and Kannan [12] reported a framework for a statistical anomaly prediction system using a Neuro-Genetic forecasting model, which predicts unauthorized invasions of user, based on previous observations and takes further action before intrusion occurs. In this paper, they have proposed an evolutionary time-series model for short-term database intrusion forecasting using genetic algorithm owing to its global search capability.

As a landmark paper in comparative study of different ANN methodologies, Samanta et al. [13] compares the performance of three types of artificial neural network (ANN), namely, multi-layer perceptron (MLP), radial basis function (RBF) network and probabilistic neural network (PNN), for bearing fault detection. Features were extracted from time domain vibration signals and they were fed as input to three popular types of ANN classifiers: namely MLP, RBF and PNN. The performance of ANNs were compared both with pre-processed and raw data.

Kumar et al. [14] touches the intrinsic complexities of the GA flow by introducing an adaptive mutation operator. The main purpose behind this approach was to improve the efficiency of GAs and to find widely distributed Pareto-optimal solutions. This algorithm was tested on some benchmark test functions and compared with other GAs. It was observed that the introduction of these mutations do improve the genetic algorithms in terms of convergence and the quality of the solutions.

In an innovative application of Neuro-genetic system, Kwon and Moon [15] proposed a hybrid Neuro-Genetic system for stock trading. A recurrent neural network (NN) having one hidden layer was used for the prediction model. The input features are generated from a number of technical indicators being used by financial experts. The genetic algorithm (GA) optimized the NN's weights under a 2-D encoding and crossover. The authors tested the proposed method with 36 companies in NYSE and NASDAQ for 13 years from 1992 to 2004. The Neuro-Genetic hybrid showed notable improvement on the average over the buy-and-hold strategy and the context-based ensemble further improved the results. They also observed that some companies were more predictable than others, which implies that the proposed Neuro-Genetic hybrid can be used for financial portfolio construction.

Ling and Leung [16] present a real-coded genetic algorithm (RCGA) with new genetic operations (crossover and mutation). They have been called the average-bound crossover and wavelet mutation. By introducing the proposed genetic operations, both the solution quality and stability are better than the RCGA with conventional genetic operations.

In the same year, Ling et al. [17] also proposed an input-dependent neural network (IDNN) with variable parameters. Since, the parameters of the neurons in the hidden nodes adapt to changes of the input environment, different test input sets separately distributed in a large domain can be tackled after training. Therefore, the author concluded that there are different individual neural networks for different sets of inputs. The proposed network exhibited a better learning and generalization ability than the traditional one.

Vizitiu et al. [18] proposed an interesting approach of feedforward neural network topology optimization based on a new fitness function definition. The experimental results in this case were compared with ones from a classic pruning method, and for their validation a proper hardware implementation of the used networks was indicated.

Tan et al. [19] put forth a feature selection scheme which selects a subset of informative attributes or variables to build models describing data, by removing redundant and irrelevant noise features. In this paper, they proposed a framework based on genetic algorithm (GA) for feature subset selection that combines various existing feature selection methods. The advantages of this approach included the ability to accommodate multiple feature selection criteria and find small subsets of features that perform well for a particular inductive learning algorithm of interest to build the classifier.

Performance results for finding the best genetic algorithm for the complex real problem of optimal machinery equipment operation and predictive maintenance were presented by Gallova [20]. The author used GA flow to modify a typical mamdanineuro-fuzzy system. The novel aspect of this paper was introduction of chaos theory principle to genetic environment for modifying parameter of neuro-fuzzy system.

Andrei Dinu et al. [21] introduced algorithm for compact neural-network hardware implementation is presented, which exploited the special properties of the Boolean functions describing the operation of artificial neurons with step activation function. The algorithm contains three steps: artificial-neural-network (ANN) mathematical model digitization, conversion of the digitized model into a logic-gate structure, and hardware optimization by elimination of redundant logic gates. This proposed strategy seems to bridge the gap between ANN design software and hardware design packages (Xilinx).

Tsmots and Skorokhoda [22] analyzed the element base for the hardware implementation of artificial neural networks (ANN). The authors have made full use of pipelining and parallel techniques and this work is one of the few attempts to select an appropriate PLD hardware for an ANN.

Haupt et al. [23] introduced a mixed intergenetic algorithm with an adaptive strategy of calculating different objective functions based upon the problem. This robust optimization technique may find application in defense and security including the case that involves the release of a chemical or biological contaminant on to a field sensor array.

Karthikeyan et al. [24] introduced a Modified Topology Crossover (MTC) scheme along with novel mutation and Genetic Operation Comlinations (GOCs) techniques. This particular scheme was applied to integer coded genetic algorithm commonly used in adhoc network. The effect of GOC on the performance of multicart routing was studied and it was found that the proposed scheme was able to achieve the two fold objective of lifetime improvement and time delay minimization.

An improved Artificial Neural Network (ANN) in the domain of optical wireless communication was presented by Yakzan et al. [25]. The network made use of a genetic algorithm-based selection and different weather models. A binary search was performed using GA and best possible model was fitted with a low bit-error-rate.

In line with the current trend of stock price forecasting using artificial intelligence, Nayak et al. [26] has used GA optimized ANN to predict daily closing price of Bombay stock exchange. They have concluded that the prediction performance of the Neuro-Genetic model is dependent on data preprocessing.

A recent paper by Al-Naqi et al. [27] presents a new adaptive algorithm that aims to control the exploration/exploitation trade-off dynamically. The algorithm is designed based on three dimensional cellular genetic algorithms (3D-cGAs). In this study, the author methodology is based on the change in the global selection pressure induced by dynamic tuning of the local selection rate. A diversity speed measure was used to guide the algorithm. A benchmark of well-known continuous test functions and real world problems was elected to investigate the effectiveness of the algorithm proposed. The results confirm that problems of various characteristics require different selection pressures, which are difficult to be identified.

Recent applications of Neuro-Genetic processing include a paper by Arabali et al. [28] which proposes a new strategy to meet the controllable heating, ventilation, and air conditioning (HVAC) load with a hybrid-renewable generation and energy storage system. Using fuzzy C-Means (FCM) clustering, experimental data are grouped into 10 clusters with similar data points to account for parametric variations. A GA-based optimization approach together with a two-point estimate method was used for minimizing the cost function.

Abdulhai et al. [29] presents a new short-term traffic flow prediction system based on an advanced Time Delay Neural Network (TDNN) model, the structure of which has been synthesized using a Genetic Algorithm (GA). The model predicts flow and occupancy values at a given freeway section based on contributions from their recent temporal profile as well the spatial profile.

CHAPTER 3

AN OVERVIEW OF GENETIC ALGORITHM

3.1 Introduction

Genetic algorithm (GA) is a search and optimization technique that is based on the evolution theory which can provide optimum solution for various problems of optimization within a relatively reasonable computation time. Genetic algorithm (GA) begins with a set of solutions represented by chromosomes, called population. Generally, a binary encoding using integer representation is used to represent each such individual. Based upon the fitness of each individual, a mating pool is created. Selection is based upon the "survival of fittest policy".

GA has a number of features:

- Genetic algorithm is a population-based search method
- GA uses recombination to mix information of candidate solutions into a new one.
- GA is stochastic.

Three most important steps in using GA are:

1. Definition of objective function
2. Definition and implementation of genetic representation
3. Definition and implementation of genetic operators

3.2 Genetic Algorithm Flow Diagram:

The basic genetic algorithm (GAs) is outlined as below:

Step I [Start] Generate random population of chromosomes, that is, suitable solutions for the problem.

Step II [Fitness] Evaluate the fitness of each chromosome in the population.

Step III [New population] Create a new population by repeating following steps until the new population is complete.

a) [Selection] Select two parent chromosomes from a population according to their fitness. Better the fitness, higher the chance to be selected to be the parent again

b) [Crossover] Crossover the parents to form new offspring. If no crossover was performed, offspring is the exact copy of parents.

c) [Mutation] mutate new offspring at each locus.

d) [Accepting] Place new offspring in the new population.

Step IV [Replace] Use new generated population for a further run of the algorithm.

Step V [Test] If the end condition is satisfied, stop, and return the best solution in current population.

Step VI [Loop] Go to step 2.

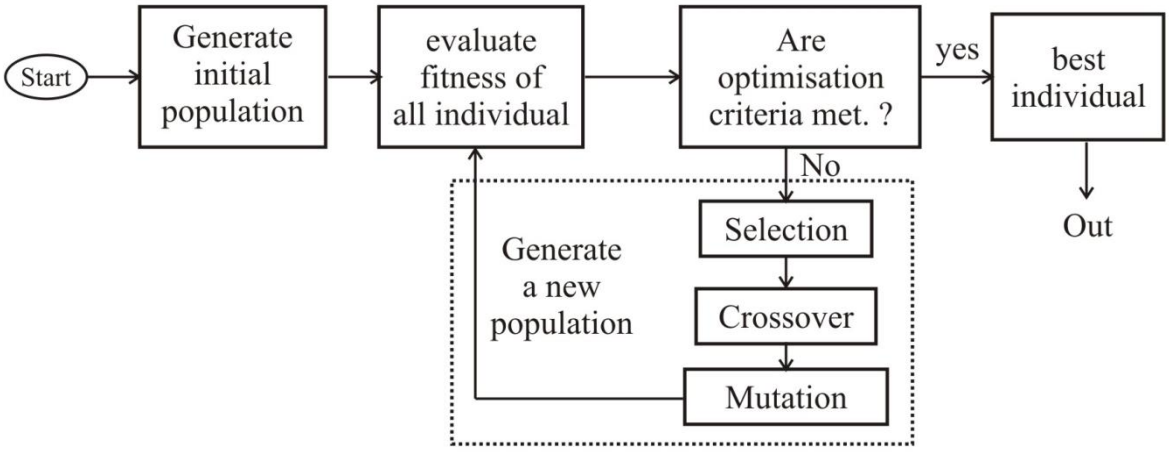

Fig. 3.1: GA Flow diagram

3.3 Fitness Function

GA is mimics the survival of the fittest principle of nature to make a search process. Therefore, GA is are naturally suitable for solving maximization problems Maximization problems are usually transformed into maximization problem by suitable transformation.

In general, a fitness function F(X) is first derived from the objective function and used in successive genetic operations. In genetic algorithm, fitness is used to allocate reproductive traits to the individuals in the population and thus act as some measure of goodness to be maximized. The following transformations are used generally to maximize or minimize an objective function.

F(X) = f(X) for maximization problem

$$F(X) = \frac{1}{(f(X))}$$ for minimization problem, if f(x) ≠ 0

$$F(X) = \frac{1}{(1+f(X))}$$, if f(X) = 0

Maximization/minimization of the objective function is followed by reproduction which is nothing but the process of creating a fitter set of offspring from already fit parents in the population.

3.4 Reproduction

Reproduction is an operator that makes more copies of better strings in a new population. Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. This is one of the reasons for the reproduction operation to be sometimes known as the selection operator. There exist a number of reproduction operators in GA literature, but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. The various method of selection chromosomes for parents to cross over are:

1. Roulette-wheel selection
2. Boltzmann selection
3. Tournament selection
4. Rank selection
5. Steady-state selection

However, the most popular techniques of selecting chromosomes are the Roulette-wheel selection and Rank Selection methods which are discussed below

3.4.1 Roulette-Wheel Selection

The commonly-used reproduction operator is the proportionate reproduction operator where a string is selected for the mating pool with a probability proportional to its fitness. Thus, the i^{th} string in the population is selected with a probability proportional to F_i. Since the population size is usually kept fixed in a simple GA, the sum of the probability of each string being selected for the mating pools must be one. Therefore, the probability for selecting the i^{th} string is

$$P_i = \frac{F_i}{\sum_{j=1}^{n} F_j} \tag{3.1}$$

where 'n' is the population size.

One way to implement this selection scheme is to imagine a roulette-wheel with its circumference marked for each string proportionate to the string's fitness as shown in Fig. (3.2). The roulette-wheel is spun 'n' times. Each time selecting an instance of the string is chosen by the Roulette-wheel pointer. Since the circumference of the wheel is marked according to a string's fitness, this roulette-wheel mechanism is expected to make $F_i/\bar{F}$ copies of the i^{th} string in the mating pool. The average fitness of the population is calculated as:

$$\overline{F} = \sum_{j=1}^{n} F_j / n \tag{3.2}$$

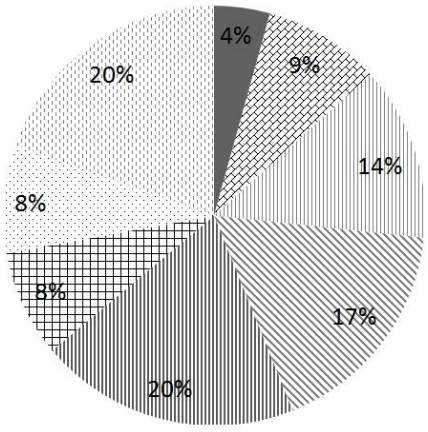

Fig. 3.2: Roulette-wheel marked for eight individual according to fitness [3].

Fig 3.2 shows a Roulette-wheel for each individual having different fitness values. The individual having higher fitness value than all other is expected to occupy more area on the roulette wheel and will be chosen more number of times to make several copies of itself. Roulette-wheel selection scheme can be simulated easily but is noisy.

Main Drawback of Roulette-wheel:

The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected.

To avoid this problem, Rank Selection method is used in our technique

3.4.2 Rank Selection

In the Rank Selection method, ranks the every chromosome of the population is ranked according to their fitness value. The best one is on the top of rank list. It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent.

3.5 Crossover

The crossover operator is the most important in GA. Crossover is a process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes

A binary variation operator is called *recombination* or *crossover*. The principle behind crossover is simple: combining two individuals with different but desirable features, we can produce an offspring which combines both of those features.

There are many kinds of crossover

a) **One-point Crossover**

The procedure of one-point crossover is to randomly generate a number (less than or equal to the chromosome length) as the crossover position. Then, keep the bits before the number unchanged and swap the bits after the crossover position between the two parents.

Example: With the two parents selected above, we randomly generate a number 2 as the crossover position:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1: 7 3| 4 5 2 2

Child 2: 1 7| 7 6 1 3

b) Two-point Crossover

The procedure of two-point crossover is similar to that of one-point crossover except that we must select two positions and only the bits between the two positions are swapped. This crossover method can preserve the first and the last parts of a chromosome and just swap the middle part.

Example: With the two parents selected above, we randomly generate two numbers 2 and 4 as the crossover positions:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1: 7 3| 4 5| 1 3

Child 2: 1 7| 7 6| 2 2

c) **Uniform Crossover**

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent.

Example: For each position, we randomly generate a number between 0 and 1, for example, 0.2, 0.7, 0.9, 0.4, 0.6, and 0.1. If the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: 7 *3 *7 6 *1 3

Parent2: 1 *7 *4 5 *2 2

Then we get two children:

Child 1: 7 7* 4* 6 2* 3

Child 2: 1 3* 7* 5 1* 2

3.6 Mutation

Mutation is a unary variation operator. It is applied to one genotype and delivers a modified *mutant*, the *child* or *offspring* of it. Similarly to crossover, mutation is a stochastic operator: the choice of what parts of each parent are combined, and the way these parts are combined, depends on random drawings.

In general, mutation is supposed to cause a random unbiased change. Mutation has a theoretical role: it can guarantee that the space is connected.

Mutation is viewed as a background operator to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, and thus ensuring ergodicity implying that there is a non-zero probability of generating any solution from any population state. However, there are a number of mutation operators reported in the literature and their choice almost always is a heuristic one. This make choice of mutation operator an effort based on experience and speculation rather than something established on sound mathematical foundations.

The next chapter on problem formulation addresses this issue and a mathematically justified mutation operator is being proposed to be incorporated into the Genetic Flow.

CHAPTER

4 THE MUTATE-DISCARD-CROSSOVER SCHEME

4.1 Problem Formulation

The literature survey presented in the previous section gives an account of many innovative Neuro-genetic techniques implemented in the recent past. However, a closer look will reveal that most of the reported techniques are application oriented and have been designed to solve a particular problem. Our endeavor however is to devise a general purpose technique, which is based on sound mathematical and statistical foundations. To achieve this objective, the primary emphasis of this work has been to improve the existing mutation and selection schemes since, mutation and selection form the core of Neuro-genetic processing flow. It was also observed during the course of study, that very few innovative mutation operators are being reported. Therefore, one of the primary concerns of this work is to devise a mutation operator which takes into account appropriateness of network weights. In this case, it is worth mentioning that in a network there are all kinds of weights including some redundant ones which do not contribute to network learning. It is required that a mutation operator should be put in place which seeks to remove those redundant weights.

Our current problem has two aspects. One is the elimination of redundant weights and other is to accomplish it using genetic algorithm. The proposed technique is described in this section.

The current task seeks to incorporate a weight pruning scheme into the conventional GA optimization flow. This has been achieved by a novel mutation scheduling scheme where only those weights are mutated which are an ideal candidate to be pruned. Conventional neural network use network pruning strategies which can be classified into two categories viz. weight decay and weight elimination. In both these strategies a complexity penalty term is defined which appropriately quantifies the degree of redundancy in a weight.

In weight decay procedure, if the initial weight vector is *w*, the complexity penalty term is given by

$$\xi_c(w) = \|w\|^2 = \sum_{i \in \zeta_{total}} {w_i}^2 \quad (4.1)$$

where, the set ζ_{total} refer to all the synaptic weights in the network.

However, in this scheme, some weights in the network have relatively larger values than others. This results in making smaller weight redundant since they have no influence on the training outcome. To overcome this limitation the complexity penalty term is now redefined as follows

$$\xi_c(w) = \sum_{i \in \zeta_{total}} \frac{(w_i/w_o)^2}{1+ (w_i/w_o)^2} \quad (4.2)$$

It is now necessary to define a parameter, which would be used for pruning of excess weights. The standard equation for a changed weight *i* at $(n+1)^{th}$ iteration is given by

$$w_i(n+1) = \frac{\left\{\frac{w_i(n)}{w_o}\right\}^2}{1+ \left\{\frac{w_i(n)}{w_o}\right\}^2} \quad (4.3)$$

where, w_o is the weight-change parameter defined according to the problem. It is proposed here that the optimality of a weight vector would be defined in the statistical sense.

Higher the complexity penalty for a particular weight, more important is the weight for the learning process.

A hidden node with an incoming weight set will contribute more to the network, if the incoming set of weights has a larger variance compared to the variance of the total weight set.

Though the problem of weight optimization using genetic algorithm has been attempted with varying degrees of success by the previous workers, it is for the first time to the best of our knowledge that the concept of weight pruning has been incorporated in the optimization process. The real challenge in this case lies in selection of an appropriate w_o.

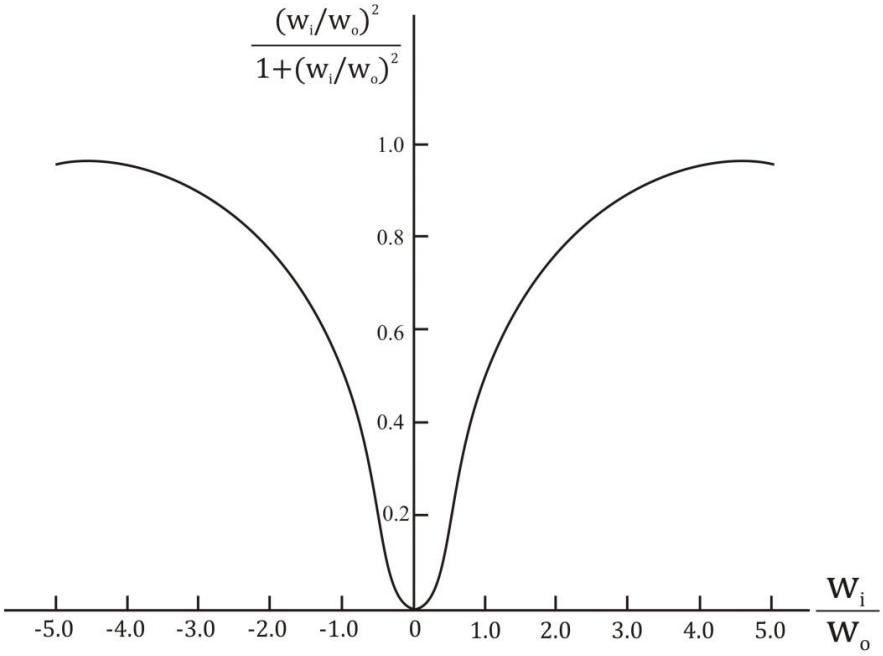

Fig. 4.1 variation of complexity penalty term with respect to w_i/w_o [2]

According to Fig. 4.1, the choice of parameter w_o is crucial aspect of network pruning. In this work, we have attempted to incorporate weight pruning strategy into the genetic flow so that weight having lesser effect on the network be rendered less fit and eventually be discarded. This strategy is now being described in next section

4.2 The Novel Mutation Operator

Let

$$h_j = \{w_{j1} + w_{j2} + w_{j3} \dots\dots\dots\dots\dots w_{ji} + b_j\} \qquad (4.4)$$

be the set of all the weights convergent on the neuron n of the hidden layer from the 1 to i^{th} inputs.

Figs 4.2 (a) and 4.3 (a) depict a neural architecture for which the mutation scheme is being described.

In order to choose an optimum set of weights, we take a look into the statistical properties of the weights converging on a hidden node.

The input to a hidden node is given by:

$$y_j = w_{ji}x_1 + w_{ji}x_2 + \cdots \ldots\ldots\ldots + w_{ji}x_i + b_j \quad (4.5)$$

where x_i is the i[th] input and is the weight from that input.

From Eq. (4.1), the mean of the input to hidden node is given by

$$E\{y_j\} = E\{\sum_{i=0}^{n} w_{ji}x_i\} = 0 \quad (4.6)$$

Since, the weights are randomly chosen, they can be considered independent of the input feature. Also, we regard the weights leading to a hidden node as zero-mean random variables.

Thus Eq. (4.6) can be modified as

$$E\{y_j\} = \sum_{i=0}^{n} E\{w_{ji}\}\, E\{x_i\} = 0 \quad (4.7)$$

Considering the second order statistics of the problem, the variance of y is given by

$$\sigma_y^2 = E\{(y)^2\} - E^2\{(y)\} \quad (4.8)$$

Using Eq. (4.6) in (4.8)

$$\sigma_y^2 = E\left\{\left(\sum_{i=0}^{n} w_{ji}x_i\right)^2\right\} \quad (4.9)$$

Or

$$\sigma_y^2 = \sum_{i,k=0}^{n} E\{(w_{ji}w_{jk}x_ix_h)\} \quad (4.10)$$

Since, the different weights loading to a hidden node as well as the weights and the input feature are independent, Eq. (4.10) becomes

$$\sigma_y^2 = \sum_{i=0}^{n} E\left\{(w_{ji})^2\right\} E\{(x_i)^2\}\ \forall\ j \quad (4.11)$$

If we normalize the training samples and make it lies within the interval [0; 1].

Thus,

$$E\{(x_i)^2\} = E\{(x_i)\}^2 + \sigma_x^2(x_i) \quad (4.12)$$

where $\sigma_x^2(x_i)$ is variance of random variable (x_i)

For a uniform random variable lying with [0, 1].

$$E\{(x_i)^2\} = \frac{1}{3} \tag{4.13}$$

Furthermore, an input-to-hidden layer weight is also regarded as a random variable with 0 mean, uniformly distributed in the interval [-a, a]. Hence, the standard deviation of the input to a hidden node is given [from Eq. (4.11) and Eq. (4.12)] by

$$\sigma_y = \sqrt{N}\frac{a}{3} \tag{4.14}$$

where N denotes the number of weights converging on a particular node.

It is evident from Eq. (4.14) that if the input to a neuron 'A' is a random variable with standard deviation σ_A, we can generate a set of weights lying in a particular range which are uniformly distributed with same standard deviation and are independent of the number of weight connected to that node. Therefore, it can be concluded that variance or standard deviation of the weight set leading to a particular node is the most important criterion for determining the information content in the randomly generated weight set.

Each node in the hidden layer is now evaluated for the effective variance of the weights converging on it. A coefficient of dominance for each node is now defined as the ratio of variance of the weights converging on it to the total variance of all the weight in the network.

$$D_{S_i} = \frac{V_{S_i}}{\bar{V}_{total}} \tag{4.15}$$

where D_{S_i} is the coefficient of dominance for the ith hidden layer neuron and $V_{S_i}$is the variance of the weights converging on node i of the hidden layer

whereas $\bar{V}_{total}$ is the variance of the set of all the weights in the hidden layer.

Similarly, coefficient of the dominance for other hidden layer neuron are calculated and a set D is defined as

$$D_i = \left\{D_{S_1}, D_{S_2}, D_{S_3} \dots \dots \dots D_{S_i}\right\} \tag{4.16}$$

The change parameter w_o is related to D as follow:

The change in the i[th] weight w_i is given by:

$$w_i = \frac{(w_i/D_i)^2}{1+(w_i/D_i)^2} \tag{4.17}$$

Since the number of weights converging on particular neuron varies with the optimized topology, we have considered here two different ANN architectures to test and compare the efficacy of proposed technique. The above mention architectures have been depicted in the Figs 4.2 (a) and 4.3 (a). The architecture of Fig. 4.2 (a) represents a classifier which each of three neurons in the output layer representing a particular class of the IRIS data described in Chapter 1. Whereas the architecture of Fig. 4.3 represent the typical function approximator whose single output neuron needs to fire at representative value corresponding to the class of each data sample.

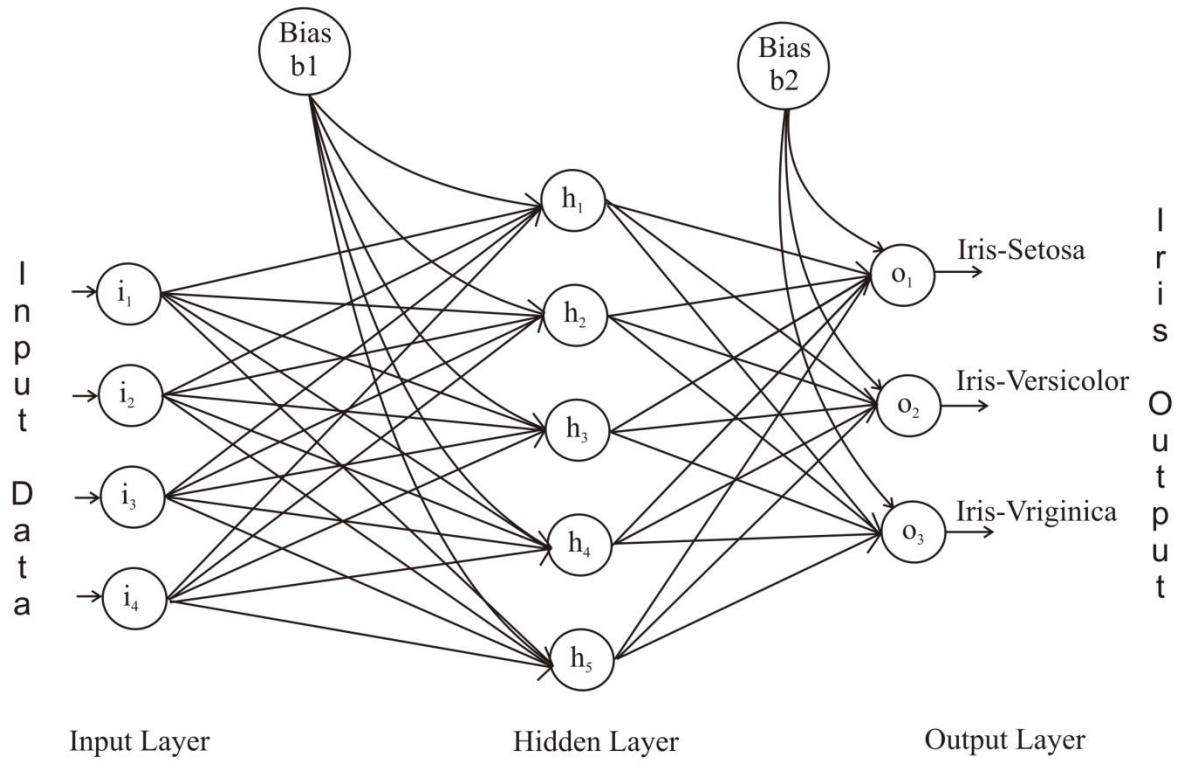

Fig. 4.2 (a) Neural Network architecture 1 for Iris classification

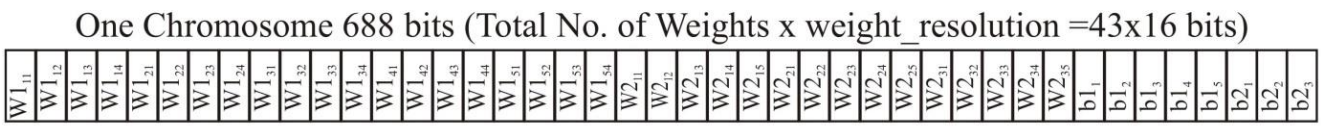

Fig. 4.2 (b) Genetic representation of ANN weights

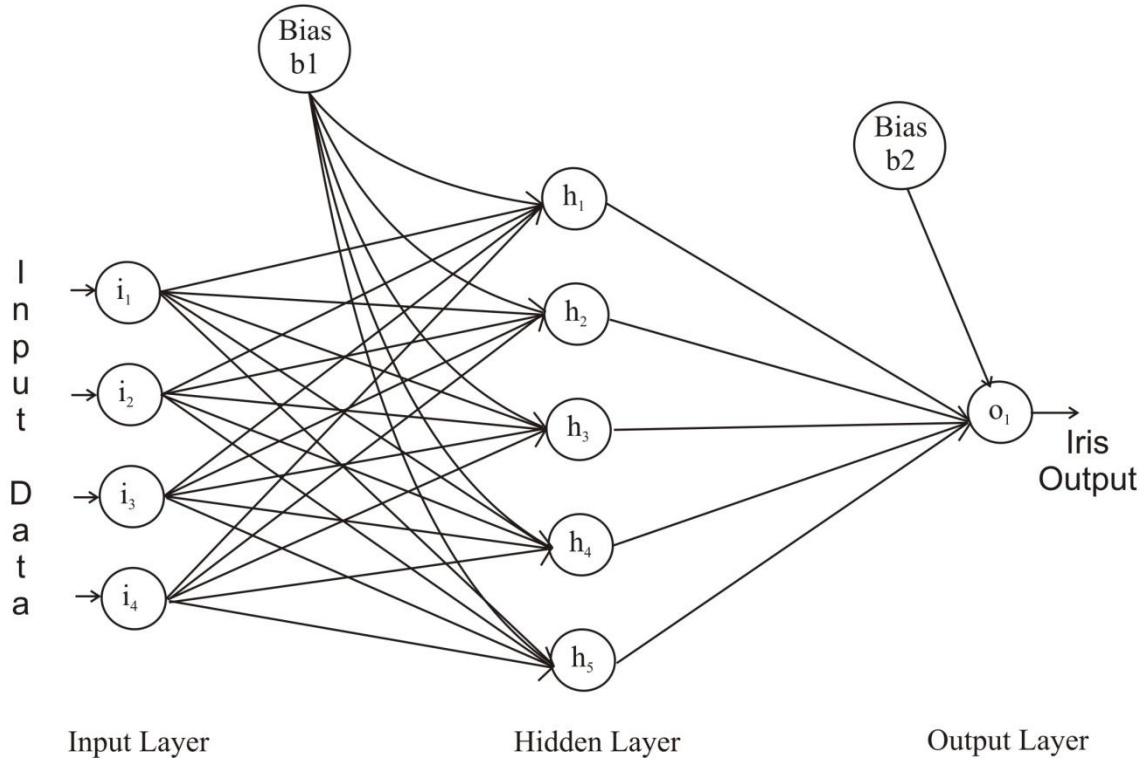

Fig. 4.3 (a) Neural Network architecture 2 for Iris classification

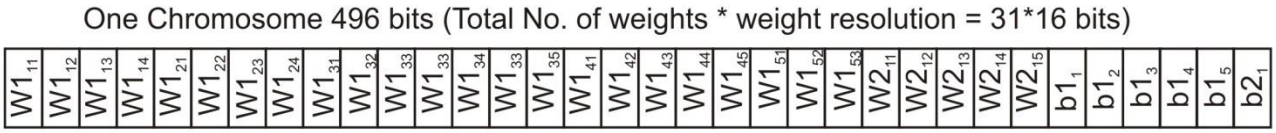

Fig. 4.3 (b) Genetic representation of ANN weights

Having formulated a novel mutation operator, it is now required to optimize a sequence of steps to be incorporated in the GA flow. Since the application of GA is mostly heuristic in nature, we take the freedom to alter the conventional flow of the algorithm according to our needs. Hence, a discard and crossover scheme which work in tandem with the mutation operator is now being described in the next section.

4.3 Mutate-Discard-Crossover Technique

The proposed "Mutate-Discard-Crossover Scheme" is depicted in the form of a flow chart shown in Fig. 4.5. Rank selection of individuals is done and the two fittest individuals take part in the crossover process to produce two offspring. However, if the offspring fitness is less than population, then the offspring are discarded otherwise it is replaced with the chromosomes having east fitness value. Now the Mutation of 3rd and 4th fittest chromosome is done according to the proposed scheme. For, the mutation process, a threshold function is defined as follow:

If mutated weight $w_i > 0.5$, then multiply weight w_i by a factor 1.5

If mutated weight $0.2 < w_i < 0.5$ than multiply weight w_i by a factor 1.0

Else, multiply weight w_i by a factor 0.2.

Again, the individuals are arranged according to their fitness values and two least fit chromosomes are discarded. Thereafter, the cross over between first two chromosomes takes place and offspring are ranked according to their fitness values. However, mutation in the next cycle again takes place for 3rd and 4th individuals and the process continues till there is no change in the fitness value of the fittest individual for a fixed number of cycles (In our case 25).

Mutipoint Restricted Crossover Operator

The crossover technique implemented here is Multipoint Restricted Crossover [5] which has been reported earlier. However, it is not a widely used method. The present requirement is to save fit weights after crossover operation at the same time increasing the fitness value of 3rd and 4th ranked individuals invoking the mutation operator. Therefore, Mutation and Multipoint Crossover (MRX) are performed in alternate iterations.

The MRX operator works as follow:

To implement the MRX operator, the ANN weights are encoded in binary string. ANN in Fig. 4.2 (a) and 4.3 (a) consist of total 43 and 31 weights respectively. These weights have been encoded in a chromosome string shown in Fig. 4.2 (b) and 4.3 (b). The number of bits to be used to represent each weight is now being decided. After some experimentation, encouraging results were obtained with 16 bits for resolution, thus giving a total chromosome length of 16 x No of weights. Crossover between two chromosomes is allowed between corresponding weights, as shown in Fig. 4.4.

It is clear from this Fig. 4.4, that randomly selected multiple crossover sites equal to the number of weights in the chromosomes are used but bit exchange is restricted to corresponding weights only. This operator is known as multiple restricted crossover or MRX. The use of MRX prevents elimination of good weights and haphazard changes in the weight values between different iterations are avoided.

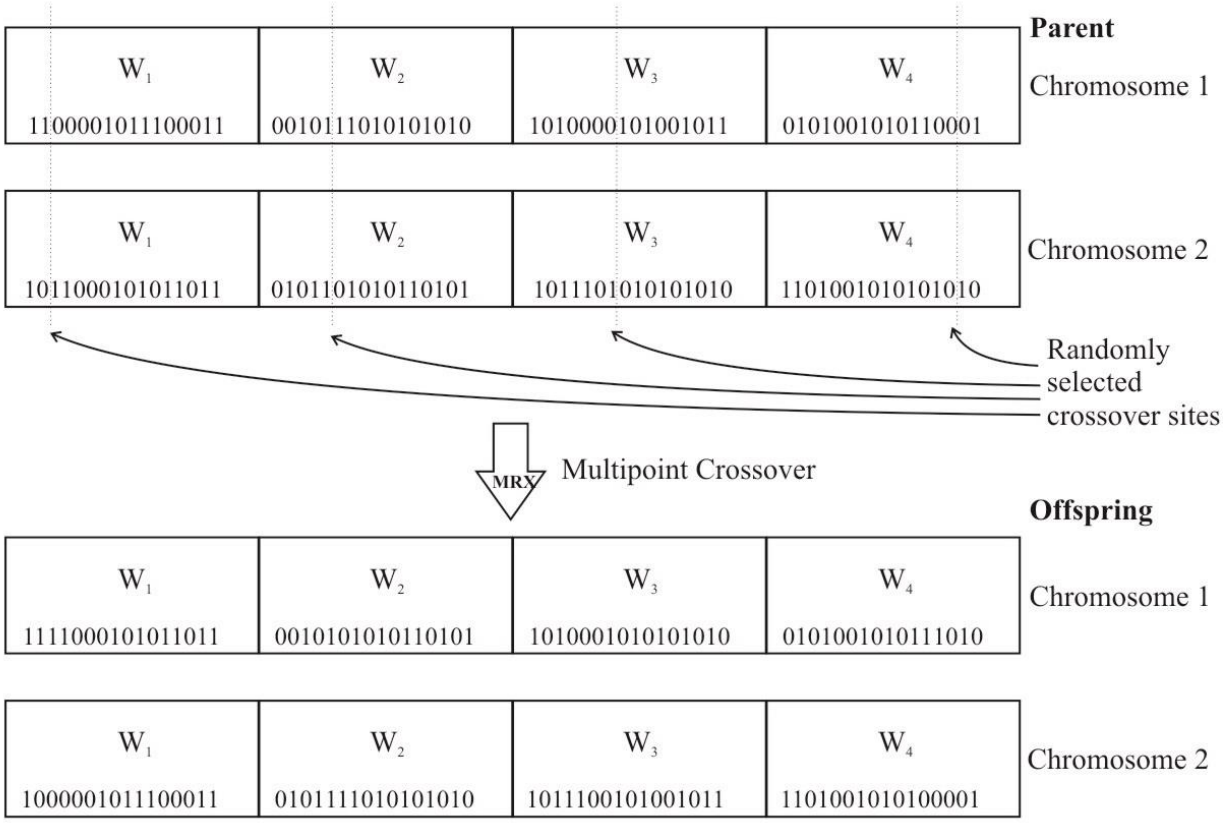

Fig. 4.4: MRX operator (for the sake of clarity only four weights are shown)

With the novel GA technique, the optimum weights for both architectures of Fig. 4.2 (a) and 4.3 (a) are obtained. Both the ANN architectures are initialized with optimized weights and the network were trained for a fixed number of epochs. The results obtained are presented and discussed in the next chapter.

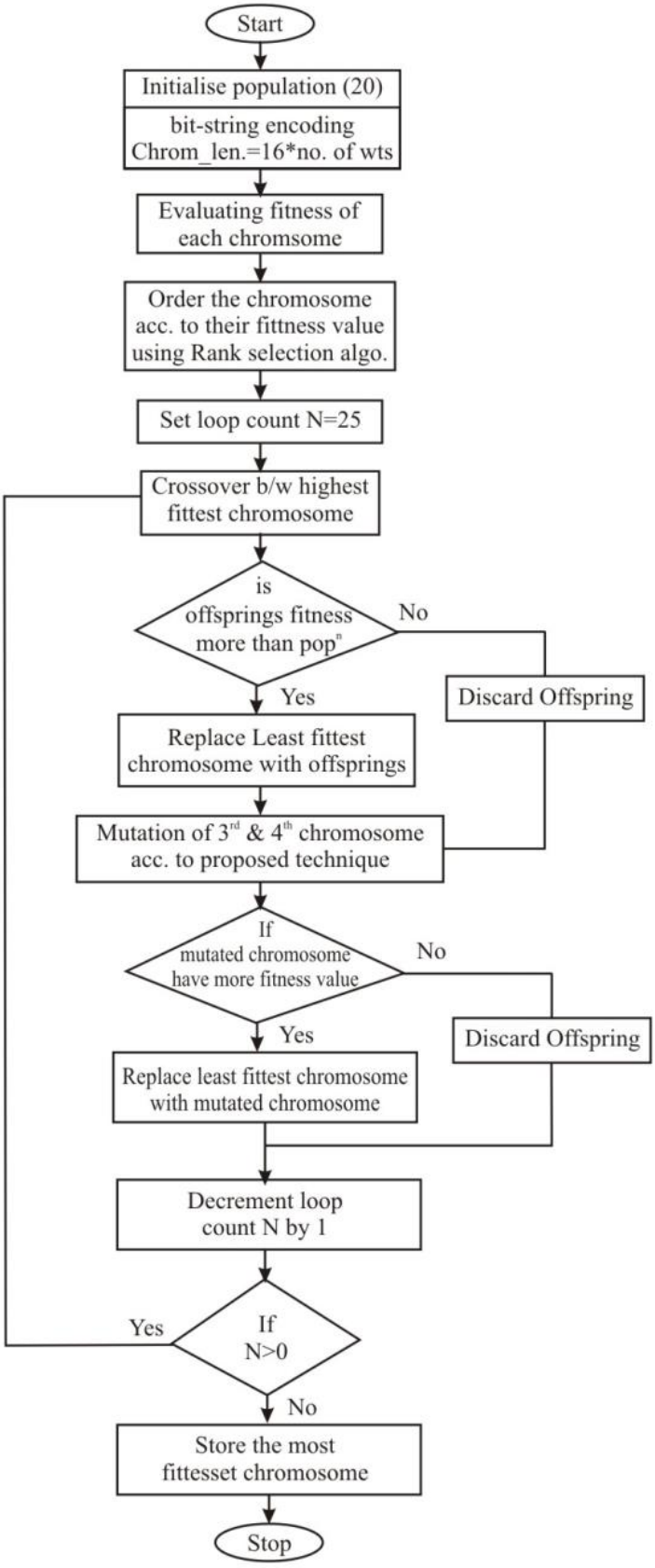

Fig. 4.5: Flow Chart for the proposed technique

CHAPTER 5

SIMULATION RESULTS AND DISCUSSION

Two different architectures were employed for classification purpose as depicted in Figs. 4.2 (a) and 4.3 (b) of the previous section. The reasons behind choosing the above mentioned architectures are as follows:

(i) The 4:5:3 architecture shown in Fig. 4.2 (a) is typical example of a classifier, where each neuron in the output layer represents a particular class. When an input sample belonging to a particular class is presented to the classifier, it required that the neuron corresponding to that particular class fires at 'one' and all other neurons are deactivated. The number of hidden layer neurons, however, has been optimized experimentally.

(ii) The 4:5:1 architecture shown in Fig. 4.3(a) is typical example of a function approximator. Here, the output neuron will fire at different values according to the class of the sample.

The performance comparison of these two architectures with randomly initialized and GA optimized weights are the work reported in this section.

All the simulations have been carried out on a computer having intel i3 processor with clock speed 2.10 GHz and 2 GB of RAM. The MATLAB version used is R2010a.
Implementation Details:

- The Iris dataset (download from UCI machine learning repository, http://archive.ics.uci.edu/ml/datasets/Iris, which is a 150x4 matrix, is taken as the input data. Out of these 150 instances, 120 instances were used for training and 30 for testing.
- The tolerance value taken was 0.01.
- Simulations were carried out for a maximum of 10000 epoch for conventional ANN and GA optimized ANN for architecture of Fig. 4.2 (a) and a maximum of 2000 epoch for architecture of Fig. 4.3 (a).

As a first step, Error surface plot have been plotted in Fig. 5.1 (a) for the architecture of Fig. 4.2 (a) when the classification was done using randomly initialized weights. The same plot for GA optimized weights in Fig. 5.1 (b) makes it evident that the error surface is now smoother with a higher spread indicating consistent performance.

Box-whisker diagrams shown in Figs. 5.2 (a) and 5.3 (b) depict the variation in the error performance at different values of momentum constant with experiments performed on five values of learning rates (viz. 0.02, 0.04, 0.06, 0.08 and 0.1) both for the randomly initialized and GA optimized weights. It can be verified that the lowest average MSE has been obtained using GA optimized weights when the momentum constant was 0.6.

Table 1 is a record of the average MSE obtained by performing 10 random experiments with randomly initialized weights at five values of learning rate (mentioned above) and nine values of momentum constant (0.1-0.9). Similarly, Table 2 summarizes the performance of the network for the same set of parameters employing GA optimized weights. Both the Tables are for the architecture of Fig. 4.2 (a). For the architecture of Fig. 4.3 (a), similar experimental data were obtained and have been represented in the form of Tables 3 and 4 for networks trained with randomly initialized and GA optimized weights respectively.

Furthermore, ten random experiments each were conducted by repeatedly initializing weights for both the architectures and the variation in error has been shown in the form of box whisker diagrams of Fig. 5.3 (a) to 5.3 (i). There was no need to draw the same diagram for GA optimized weights as they were optimized using a fixed population. However, as mentioned in the previous paragraph, the average MSE has been lower than that obtained with conventional ANNs.

Another remarkable thing can be observed from the box whisker diagrams in the lower error values obtained with architecture of Fig. 4.3 (a). The surface plots shown in Figs. 5.4 (a) & (b) for the second architecture also prove the consistent performance of GA optimized weight sets.

Finally, tables 3 to 6 summarize percentage classification results of the IRIS data for architecture in Fig 4.2 (a) and tables 9 to 12 depict the same for architecture in Fig 4.3 (a)

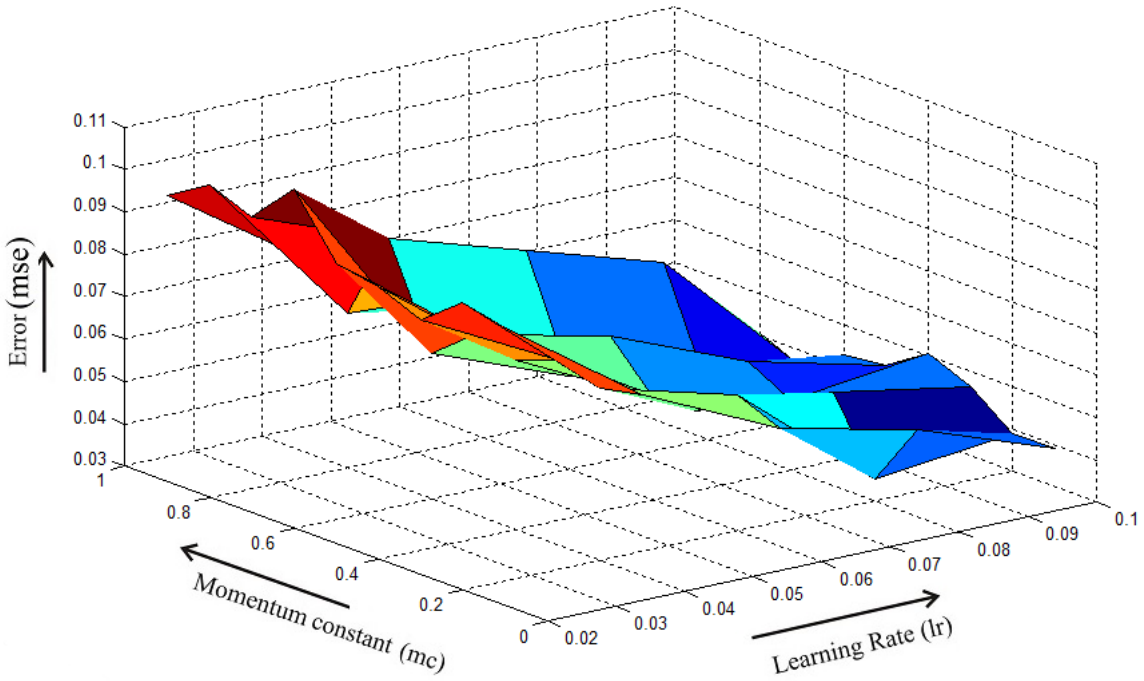

Fig. 5.1 (a): Error Surface plot for conventional back-propagation ANN for architecture of Fig. 4.2 (a)

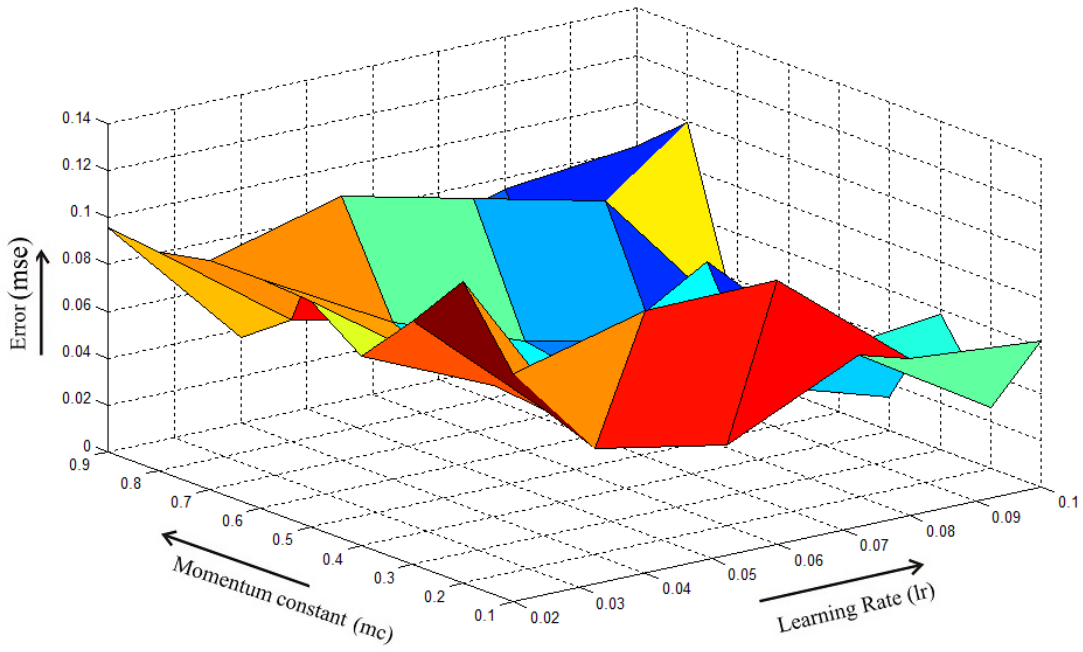

Fig. 5.1 (b): Error Surface plot for ANN trained with GA optimized weights for architecture of Fig. 4.2 (a)

Table 1: Error (mse) after 10000 Epochs for conventional back-propagation ANN for architecture of Fig. 4.2 (a)

↓η; α→	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.02	0.09541	0.09799	0.09008	0.09446	0.0959	0.10994	0.09986	0.10355	0.09744
0.04	0.07289	0.07079	0.0737	0.06989	0.07222	0.06426	0.08754	0.06624	0.07558
0.06	0.05772	0.06222	0.05466	0.05404	0.06481	0.05183	0.07787	0.06483	0.05659
0.08	0.05062	0.03527	0.05202	0.04655	0.05178	0.04255	0.06804	0.04947	0.05909
0.1	0.03904	0.03876	0.04633	0.05041	0.04382	0.0429	0.03637	0.04052	0.0396

Table 2: Error (mse) after 1000 Epochs for ANN trained with GA optimized weights for architecture of Fig. 4.2 (a)

↓η; α→	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.02	0.0975	0.129	0.104	0.0807	0.0965	0.0976	0.0976	0.093	0.0957
0.04	0.112	0.0455	0.0536	0.056	0.0564	0.0672	0.0672	0.0519	0.0365
0.06	0.113	0.0345	0.042	0.0566	0.0427	0.0467	0.0467	0.0392	0.059
0.08	0.0669	0.0604	0.0512	0.0314	0.0762	0.0339	0.0864	0.0322	0.0753
0.1	0.0621	0.0256	0.0576	0.0144	0.0231	0.0313	0.0222	0.0995	0.0815

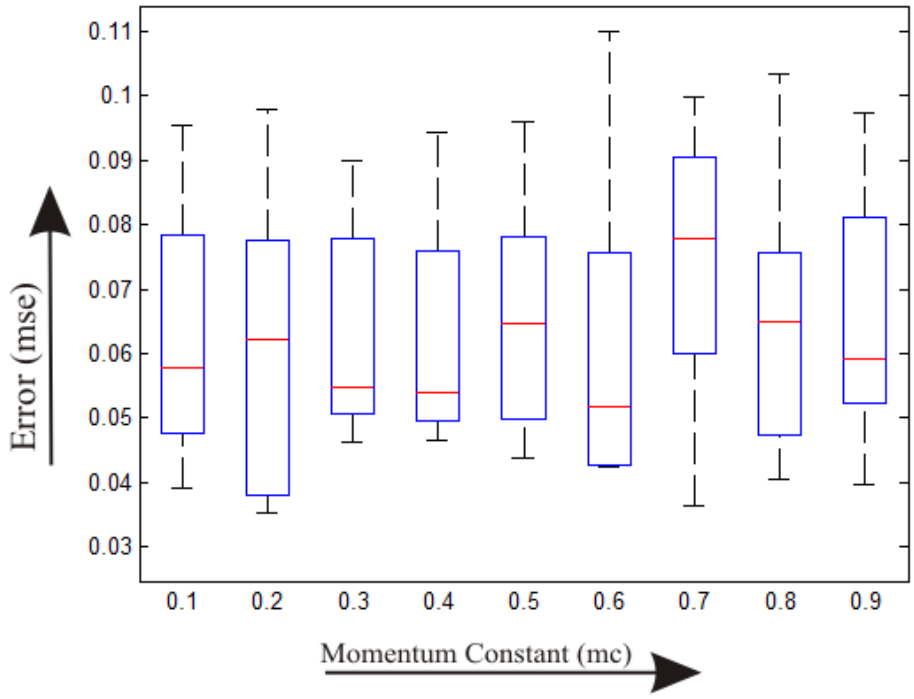

Fig. 5.2(a): Box whisker diagram for the error obtained using architecture of Fig. 4.2 (a) with randomly initialized weights

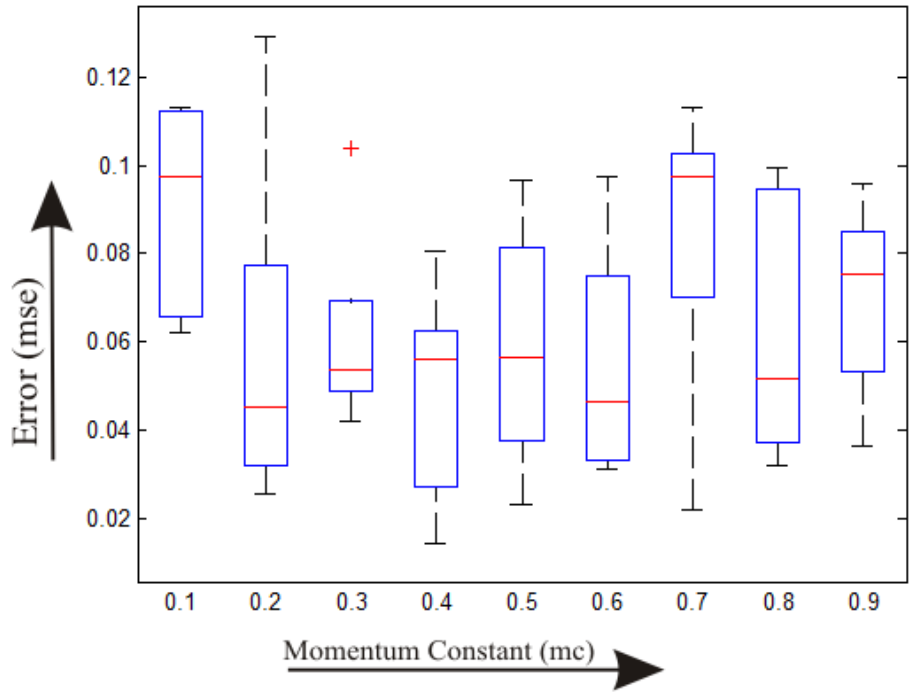

Fig. 5.2 (b): Box whisker diagram for the error obtained using architecture of Fig. 4.2 (a) with GA optimized weights

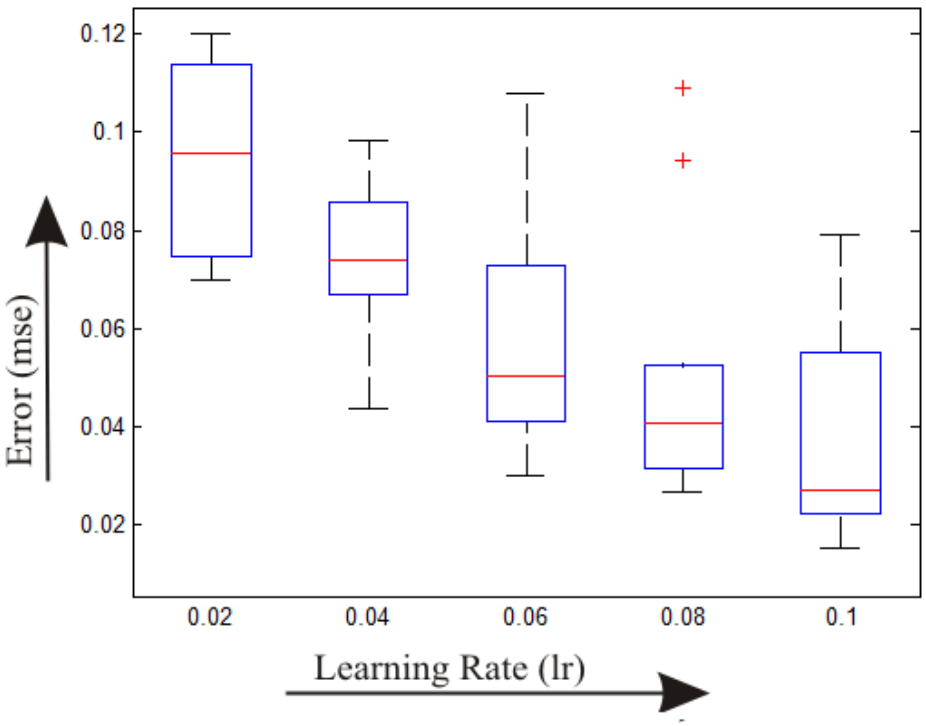

Fig. 5.3 (a): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.1

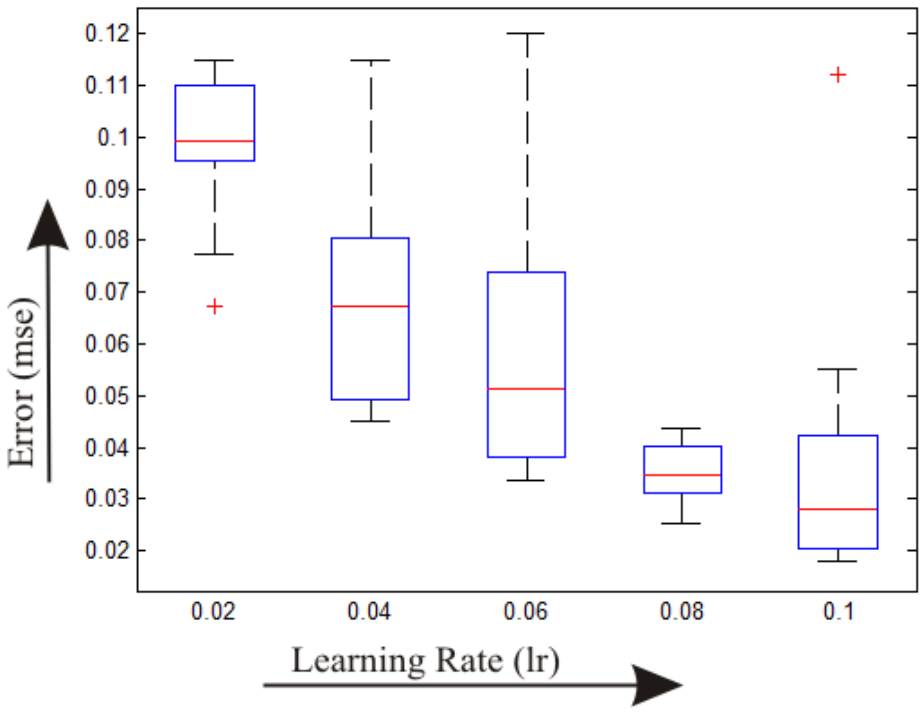

Fig. 5.3 (b): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.2

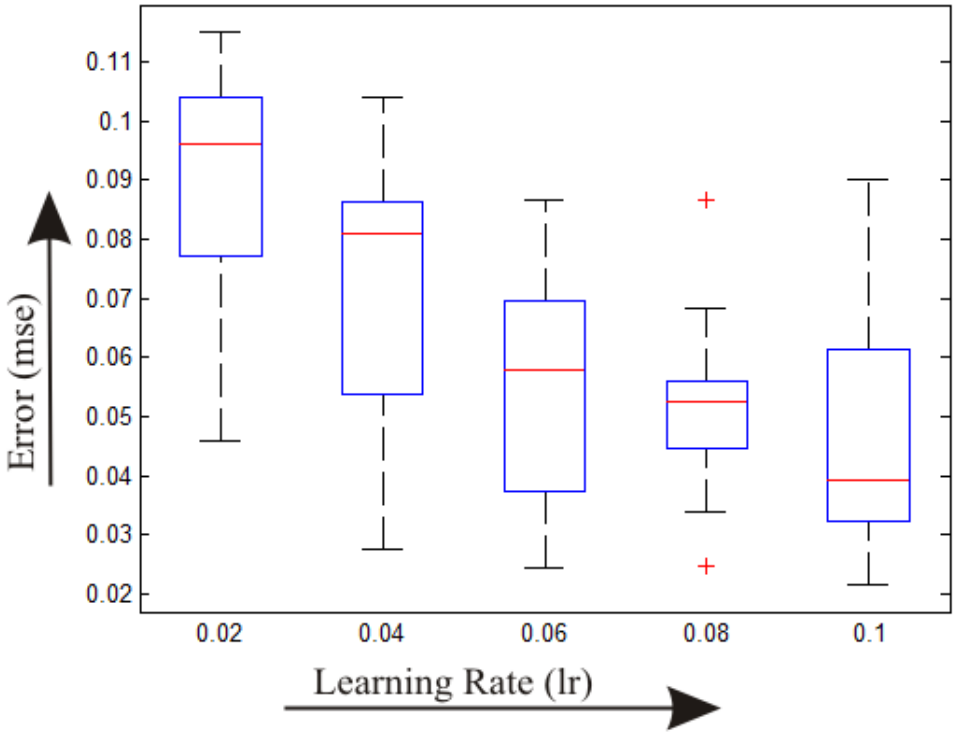

Fig. 5.3 (c): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.3

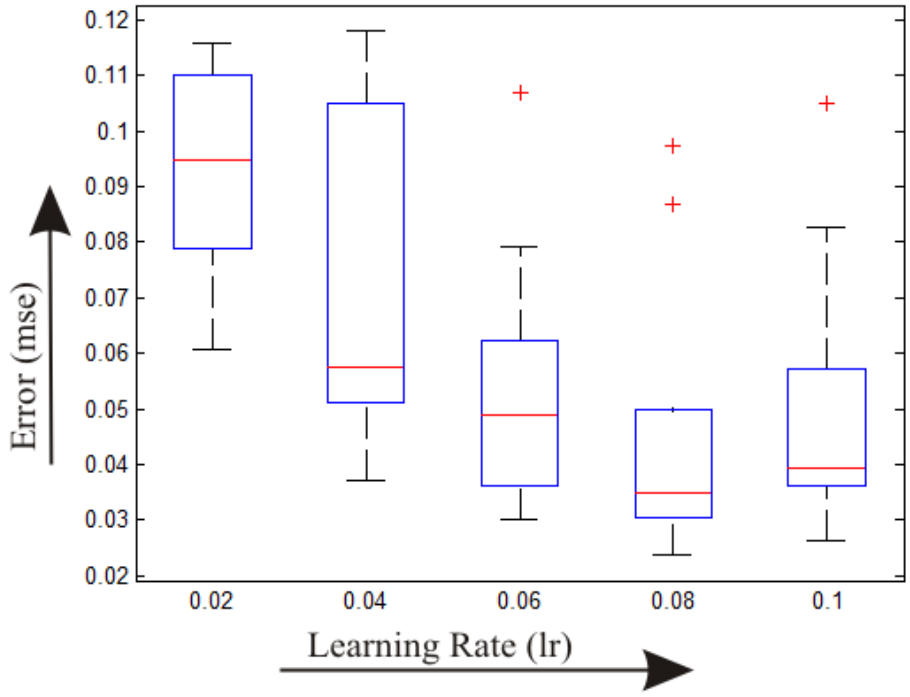

Fig. 5.3 (d): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.4

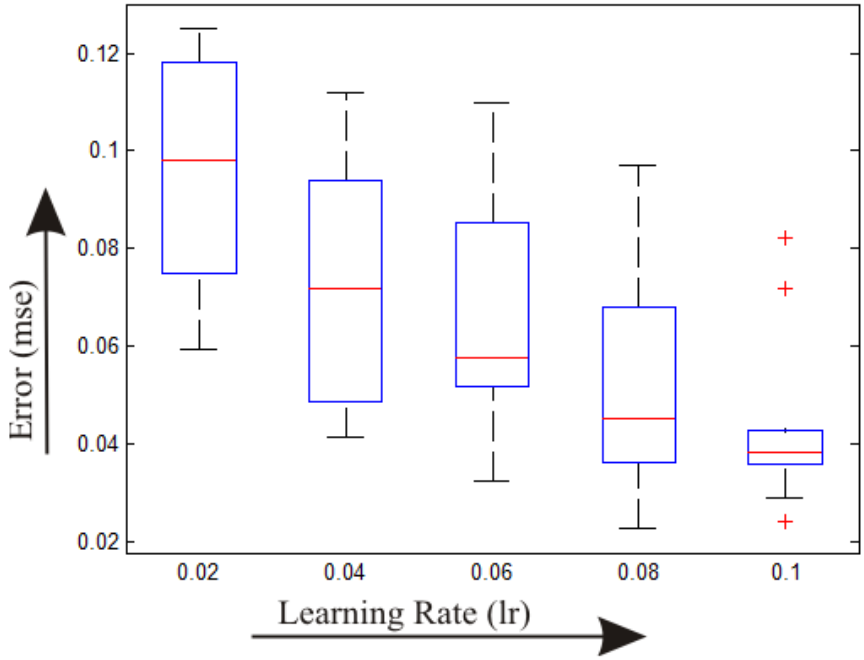

Fig. 5.3 (e): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.5

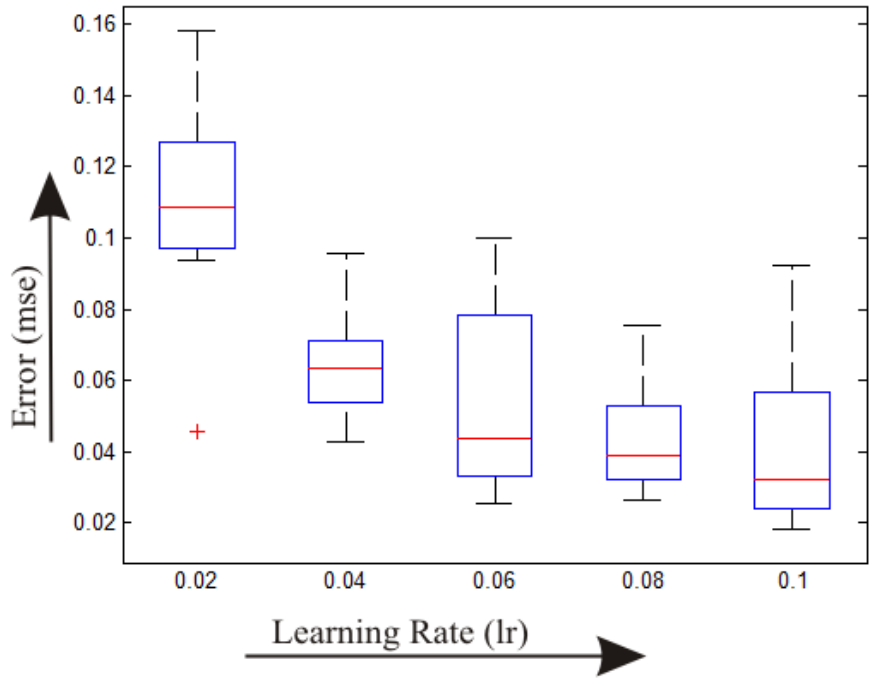

Fig. 5.3 (f): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.6

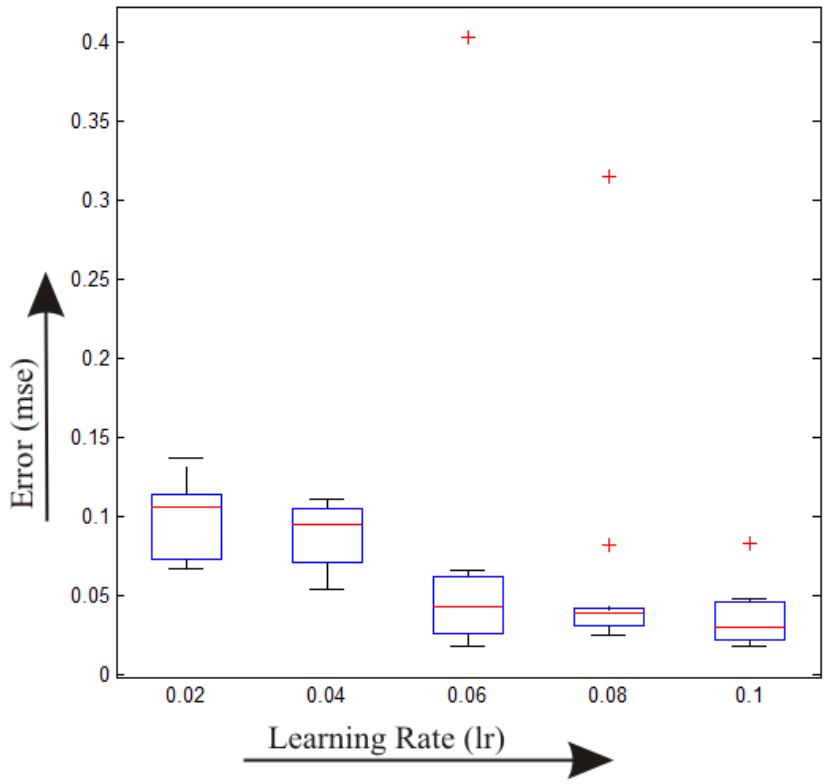

Fig. 5.3 (g): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.7

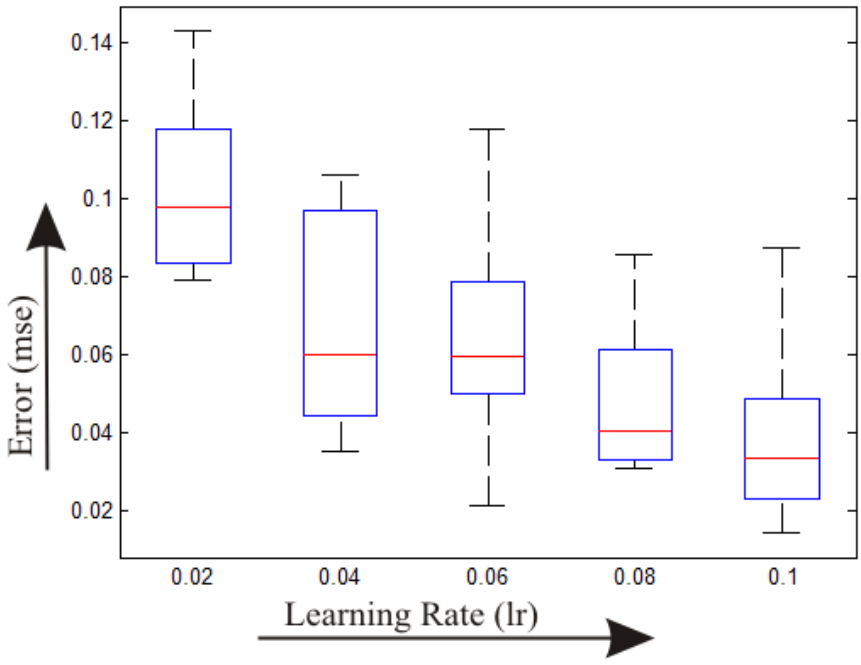

Fig. 5.3 (h): Box whisker diagram for the error obtained using architecture of Fig. 4.2(a) with randomly initialized weights at mc (α) = 0.8

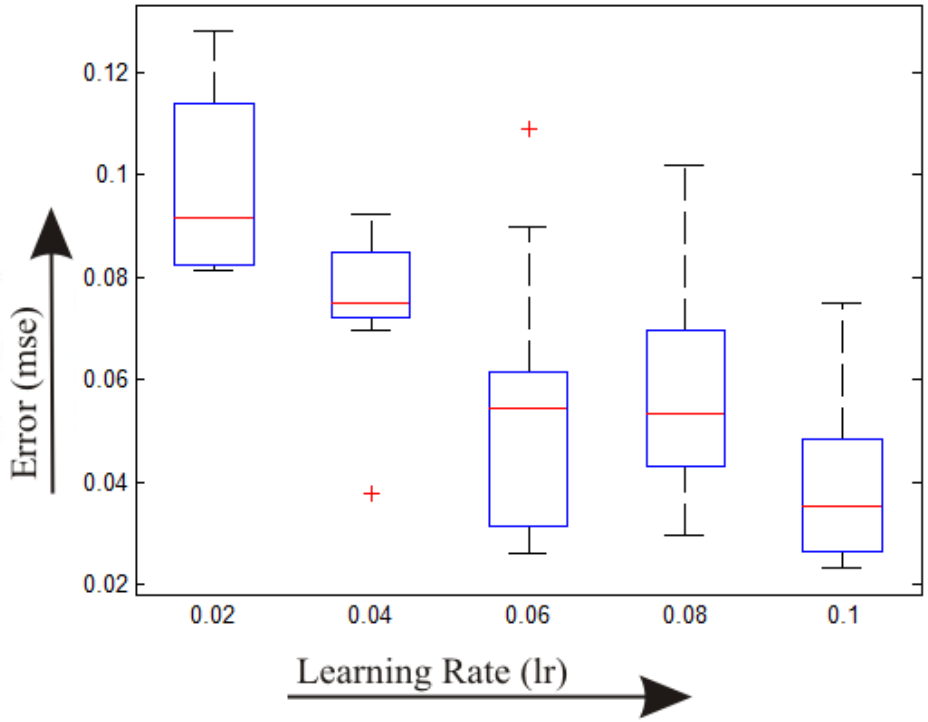

Fig. 5.3 (i): Box whisker diagram for the error obtained using architecture of Fig. 4.2 (a) with randomly initialized weights at mc (α) = 0.9

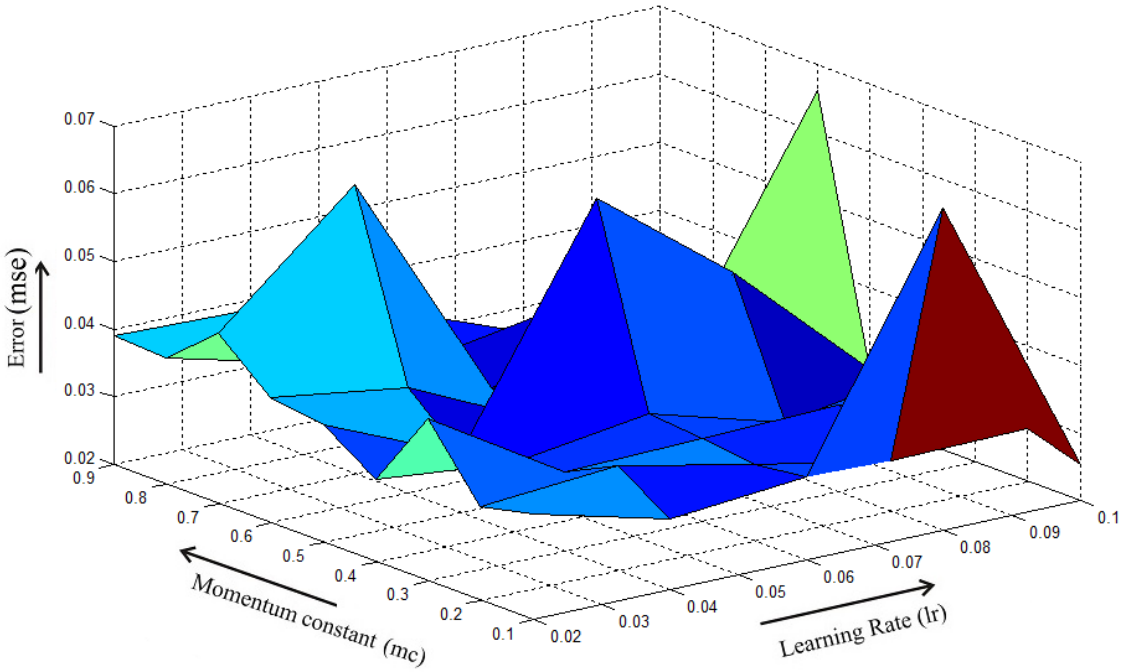

Fig. 5.4 (a): Error Surface plot for conventional back-propagation ANN for architecture of Fig. 4.3 (a)

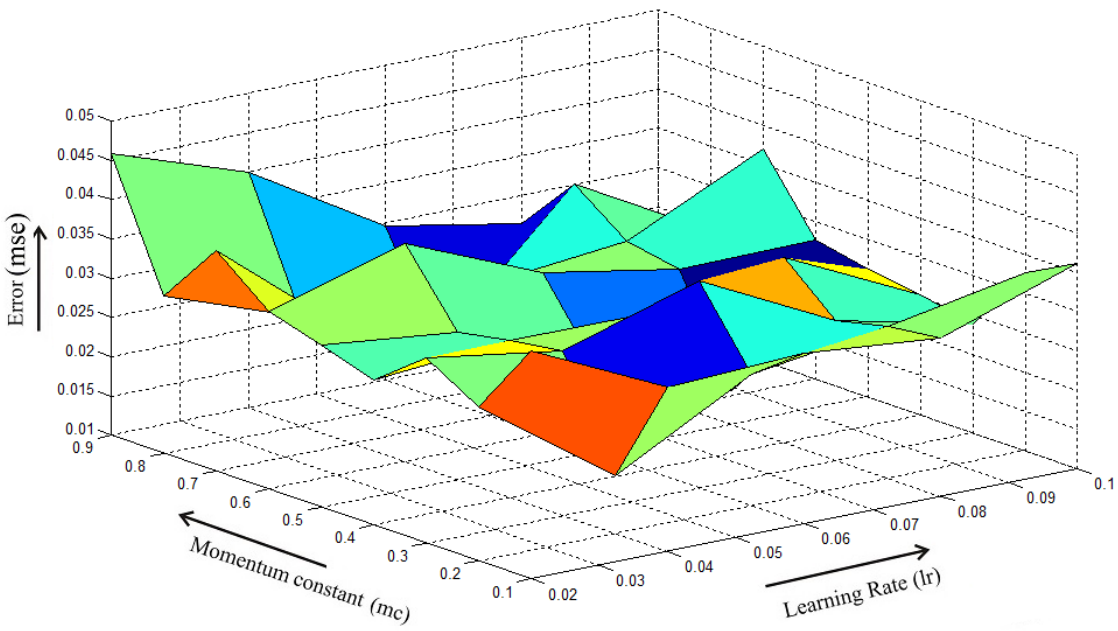

Fig. 5.4 (b): Error Surface plot for ANN trained with GA optimized weights for architecture of Fig. 4.3 (a)

Table 3: Error (mse) after 2000 Epochs for conventional back-propagation ANN for architecture of Fig. 4.3 (a)

$\downarrow\eta; \alpha\rightarrow$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.02	0.03559	0.03381	0.04394	0.0322	0.03725	0.03846	0.0451	0.03867	0.03905
0.04	0.03054	0.03548	0.03168	0.03011	0.02874	0.03556	0.06263	0.03347	0.03844
0.06	0.0323	0.03123	0.03221	0.03313	0.062	0.02626	0.02835	0.03047	0.03406
0.08	0.06751	0.02743	0.03285	0.0274	0.04665	0.0329	0.03174	0.03383	0.02644
0.1	0.02533	0.0278	0.02481	0.03089	0.028	0.06634	0.02573	0.02678	0.03257

Table 4: Error (mse) after 1000 Epochs for ANN trained with GA optimized weights for architecture of Fig. 4.3 (a)

$\downarrow\eta; \alpha\rightarrow$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.02	0.0391	0.0297	0.0336	0.0284	0.0306	0.0327	0.0381	0.03	0.0458
0.04	0.0309	0.0173	0.031	0.03	0.0288	0.0378	0.024	0.0238	0.0399
0.06	0.0317	0.0268	0.0363	0.0299	0.0213	0.0305	0.0269	0.0168	0.0296
0.08	0.0301	0.0294	0.0278	0.0336	0.0143	0.0275	0.0287	0.0337	0.0264
0.1	0.0362	0.0326	0.0238	0.0252	0.0262	0.0277	0.0369	0.0243	0.0232

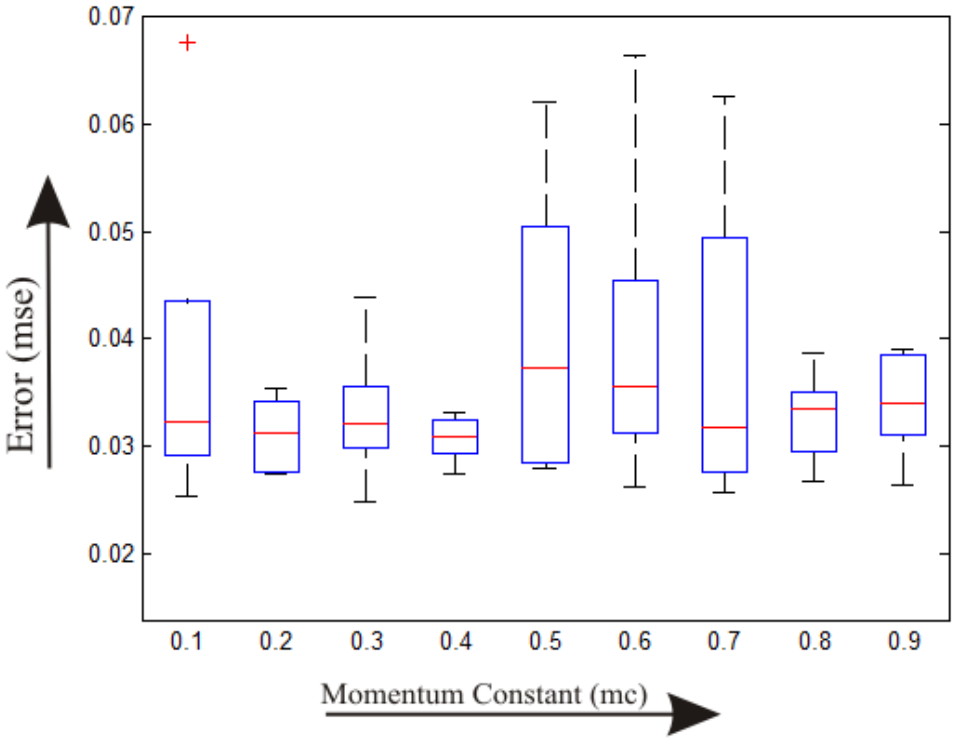

Fig. 5.5(a): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights

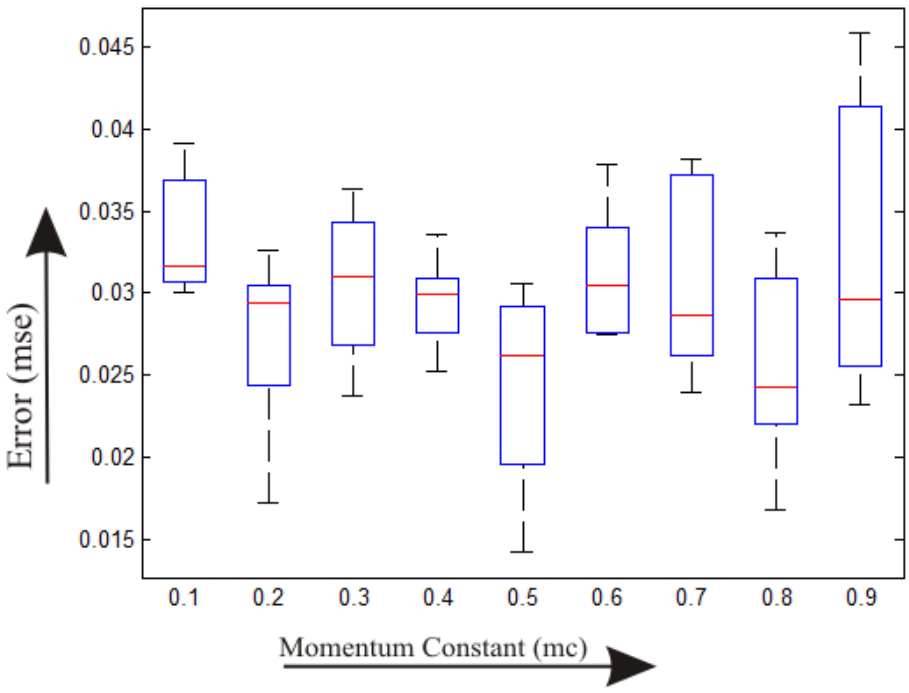

Fig. 5.5 (b): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with GA optimized weights

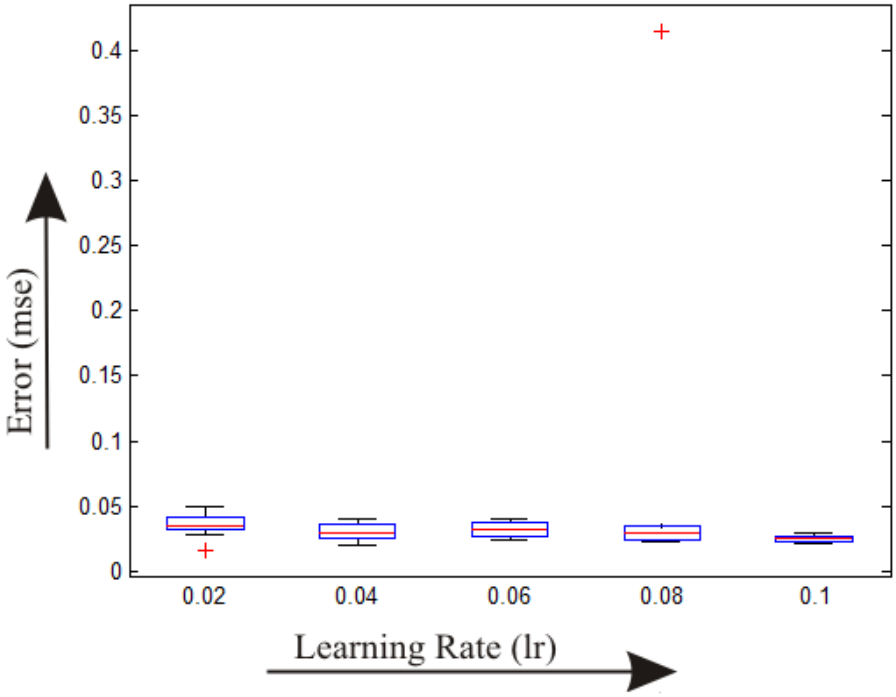

Fig. 5.6 (a): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.1

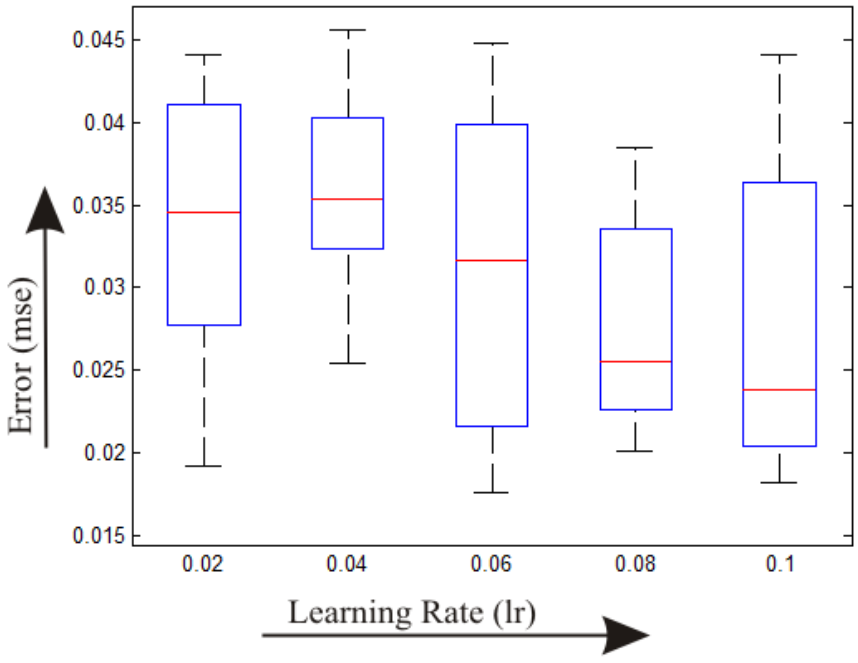

Fig. 5.6 (b): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.2

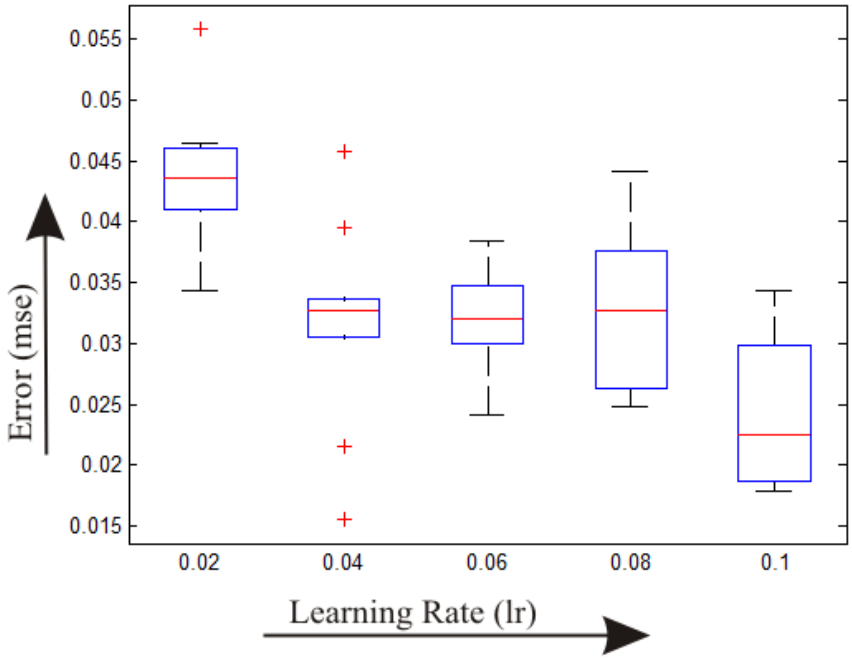

Fig. 5.6 (c): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.3

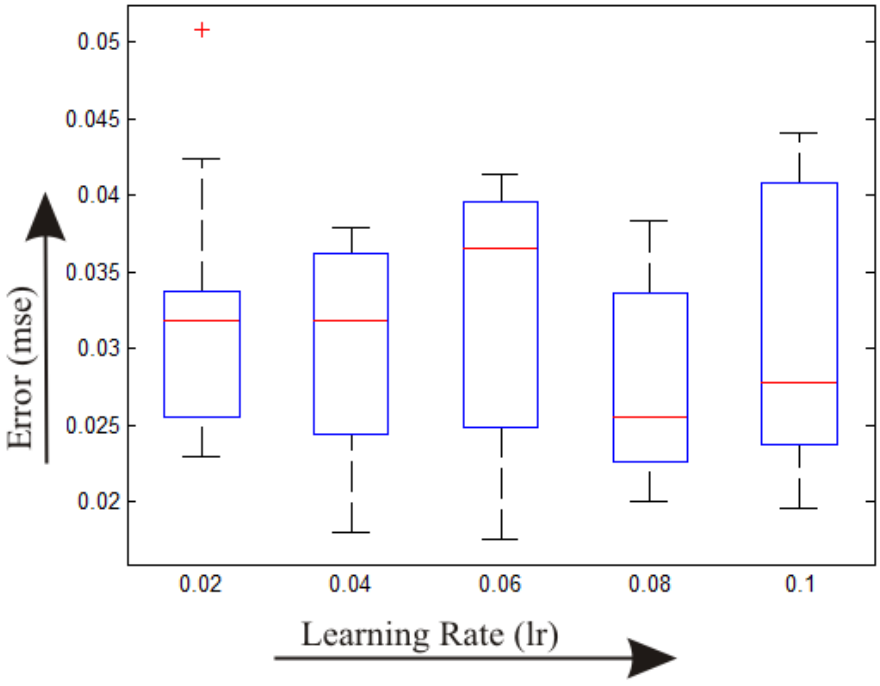

Fig. 5.6 (d): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.4

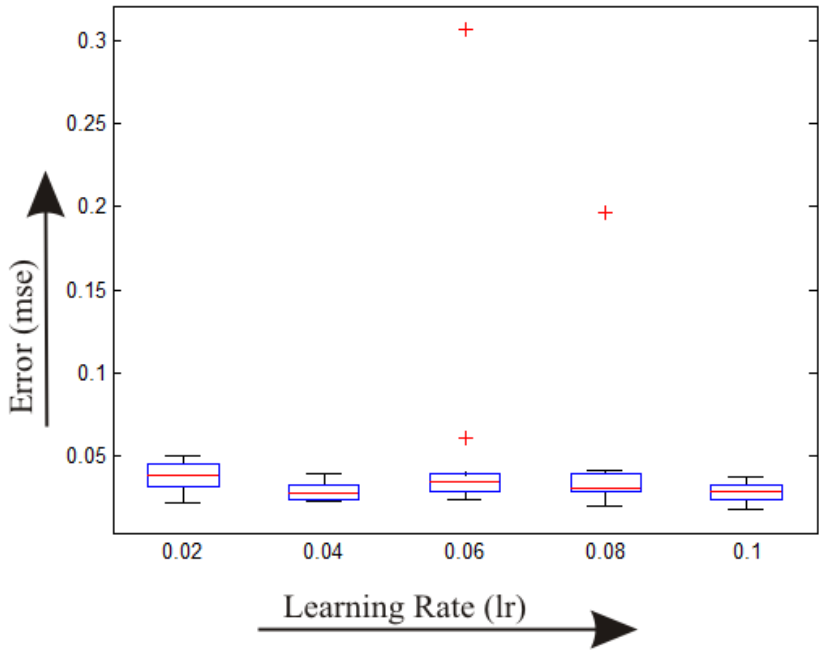

Fig. 5.6 (e): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.5

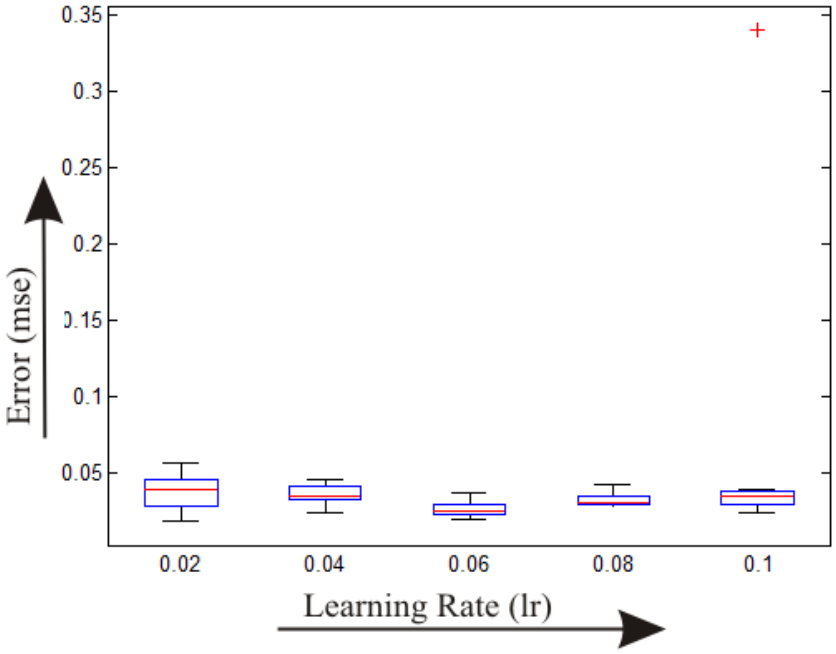

Fig. 5.6 (f): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.6

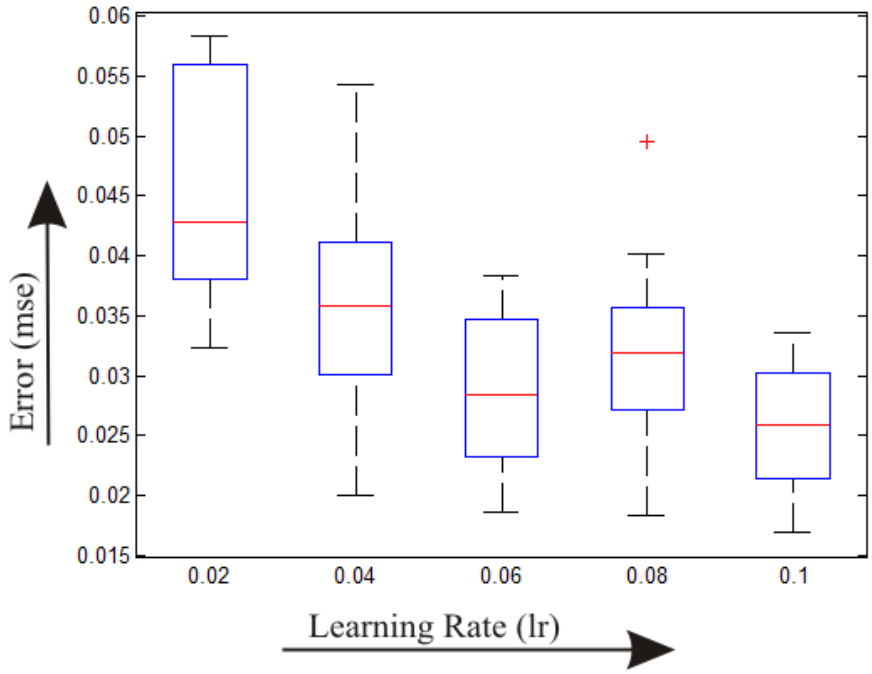

Fig. 5.6 (g): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.7

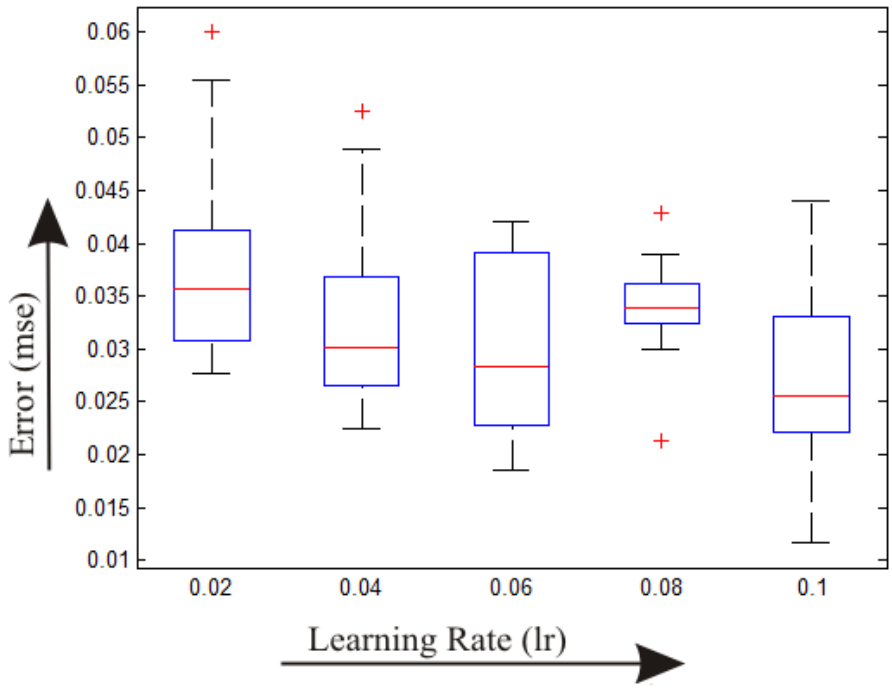

Fig. 5.6 (h): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.8

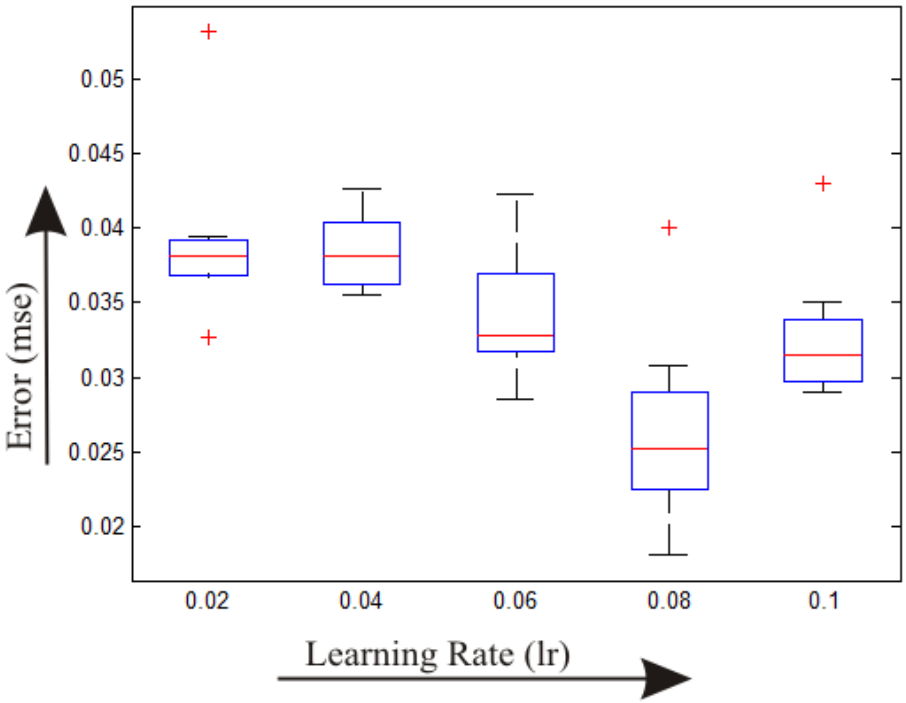

Fig. 5.6 (i): Box whisker diagram for the error obtained using architecture of Fig. 4.3 (a) with randomly initialized weights at mc (α) = 0.9

Table 5: Result on Training Data (120 samples) at α=0.7 & η=0.1 for architecture of Fig. 4.2 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	99.16%	80.83%	84.16%
Conventional ANN with random weights at 10000 epochs	100%	90.8%	93.33%

Table 6: Result on Training Data (120 samples) at α=0.2 & η=0.08 for architecture of Fig. 4.2 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	87.5%	80%	83.33%
Conventional ANN with random weights at 10000 epochs	100%	88.33%	85%

Table 7: Result on Test Data (30 samples) at α=0.7 & η=0.1 for architecture of Fig. 4.2 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	93.33%	93.33%	96.67%
Conventional ANN with random weights at 10000 epochs	96.67%	93.33%	96.67%

Table 8: Result on Test Data (30 samples) at α=0.2 & η=0.08 for architecture of Fig. 4.2 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	96.67%	90%	76.67%
Conventional ANN with random weights at 10000 epochs	100%	96.67%	86.67%

Table 9: Result on Training Data (120 samples) at α=0.7 & η=0.1 for architecture of Fig. 4.3 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	100%	72.5%	72.5%
Conventional ANN with random weights at 10000 epochs	97.5%	85%	77.5%

Table 10: Result on Training Data (120 samples) at α=0.2 & η=0.08 for architecture of Fig. 4.3 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	100%	65%	82.5
Conventional ANN with random weights at 10000 epochs	100%	87.5%	70%

Table 11: Result on Test Data (30 samples) at α=0.7 & η=0.1 for architecture of Fig. 4.3 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	100%	90%	80%
Conventional ANN with random weights at 2000 epochs	90%	90%	90%

Table 12: Result on Training Data (30 samples) at α=0.2 & η=0.08 for architecture of Fig. 4.3 (a)

% Classification	Iris Classes		
	Setosa	Versicolor	Virginica
With GA determined weights at 1000 epochs	100%	80%	70%
Conventional ANN with random weights at 2000 epochs	100%	70%	90%

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The optimization of weights for an ANN trained with backpropagation algorithm using genetic algorithm is an important step towards achieving hardware parallelism since, the computation of local gradient for each neuron in the network is an intensive task taking several machine cycles to complete. In contrast GA flow, inspite of being iterative, is simple in nature and once the weights are obtained, the reduction in number of epochs compensates for the GA calculations. It can safely be concluded that future VLSI implementation of ANN will contain a separate GA module for pre-processing and weights optimization. Based upon the present work and simulation results reported in the previous chapter, the following conclusion can be drawn:

1. A statistically justified mutation operator that takes into account variance if incoming weights is an effective method to incorporate weights pruning into genetic flow.
2. Mutation and crossover when performed in tandem help to retain the fit weights and increase the possibility of near weights to consolidate their position.
3. The ANN when trained using weights obtained from the proposed technique gives comparable performance in lesser number of epochs
4. The performance of network with GA optimized weights has been more consistent in terms of MSE obtained and classification performed.

The present study is a stepping stone to custom hardware implementation of an integrated Neuro-Genetic processor. In future, the GA-processor can be realized as a software module, which serves as a pre-processor to the hardware ANN using microcontroller or hardware implementation of GA itself can be attempted.

PUBLICATIONS

1. Ravi Kumar and Sunil Kumar, "A Neuro-Genetic classifier with a weight pruning based mutation operator and a modified selection scheme", *Soft Computing, Springer*. (Communicated)

2. Sunil Kumar and Ravi Kumar, "A Mutate-Discard-Crossover Scheme for Genetic Optimization of ANN Weights", *IETE Journal of Research.* (Communicated)

3. Sunil Kumar and Ravi Kumar, "Performance Analysis of Single-Point and Multipoint Restricted Crossover Schemes in the Design of a Neuro-Genetic Pattern Classifier" *International Conference on Electronics, Communication and Information Technology (ICECIT), 4-5 Oct 2013, Thapar University, Patiala (*Accepted)

REFERENCES

[1] Simon Haykin, *Neural Networks A Comprehensive Foundation*, Canada: Prentice Hall, 1999, pp. 23-66, 72-90,

[2] Simon Hykin, *Neural network and learning machines*, India Prentice-HALL, 2007, chapter 4.

[3] S.N. Sivanandam and S.N. Deepa, *Introduction to Genetic Algorithms*, Springer-Verlag, Berlin Heidelberg, 2008, pp. 15-60.

[4] Minoru Fukumi and Norio Akamatsu, "A Method to Design a Neural Pattern Recognition System by Using a Genetic Algorithm with Partial Fitness and a Deterministic Mutation," *IEEE International Conference on Systems*, Man and Cybernetics, vol. 3, pp. 1989-1993, 14-17 Oct. 1996.

[5] A.K. Srivastava, K.K. Shukla and S.K. Srivastava, "Exploring Neuro-Genetic Processing of Electronic Nose Data," *Microelectronics Journal*, vol. 29, pp. 921-931, 1998.

[6] A. Hunter and K.S. Chiu, "Genetic Algorithm Design of Neural Network and Fuzzy Logic Controllers," *Soft Comput*, vol. 4, pp. 186-192, 2000.

[7] S. Marsili Libelli and P. Alba, "Adaptive Mutation in Genetic Algorithms," *Soft Comput*, vol. 4, pp. 76-80, 2000.

[8] L. Bull, "On Coevolutionary Genetic Algorithms," *Soft Comput*, vol. 5, pp. 201-207, 2000.

[9] Tetsuya Imai, Masaya Yoshikawa, Hidekazu Terai and H. Yamauchi, "Scalable GA Processor Architecture and its Implementation of Processor-Element," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 3, pp. 3148-3151, 2002.

[10] P. Kalra and Neelam Rup Prakash, "A Neuro-Genetic Algorithm Approach for Solving the Inverse Kinematics of Robotic Manipulators," *IEEE Conference on Systems, Man and Cybernetics*, vol. 2, pp. 1979-1984, 5-8 Oct 2003.

[11] S. J. Louis, "Genetic Learning for Combinational Logic Design," *Soft Comput*, vol. 9, pp. 38-43, 2005.

[12] P. Ramasubramanian and A. Kannan, "A Genetic-Algorithm Based Neural Network Short-term fore Casting Framework for Database Intrusion Prediction System," *Soft Comput*, vol. 10, pp. 699-714, 2006.

[13] B. Samanta, K.R. Al-Balushi and S.A. Al-Araimi, "Artificial Neural Networks and Genetic Algorithm for Bearing Fault Detection," *Soft Comput*, vol. 10, pp. 264-271, 2006.

[14] P. Kumar, D. Gospodaric and P. Bauer, "Improved Genetic Algorithm Inspired by Biological Evolution," *Soft Comput*, vol. 11, pp. 923–941, 2007.

[15] Yung-Keun Kwon and Byung-Ro Moon, "A Hybrid Neuro-Genetic approach for Stock Forecasting," *IEEE Trans. on Neural Network*, vol. 18, no. 3, pp. 851-864, May 2007.

[16] S. H. Ling and F.H.F. Leung, "An Improved Genetic Algorithm with Average-Bound Crossover and Wavelet Mutation Operations," *Soft Comput*, vol. 11, pp. 7–31, 2007.

[17] S. H. Ling, F.H.F. Leung and H. K. Lam, "Input-Dependent Neural Network Trained by Real-Coded Genetic Algorithm and its Industrial Applications," *Soft Comput*, vol. 11, pp. 1033–1052, 2007.

[18] C. I. Vizitiu, A. Radu, T. Oroian and C. Molder, "Optimal Hardware Implementation of a Feedforward Neural Network Topology using a Genetic Algorithm For Prunning," *IEEE conference on Semiconductor (CAS 2007)*, pp. 459-462, 17 Sept.-15 Oct. 2007.

[19] Feng Tan, Xuezheng Fu, Yanqing Zhang and Anu G. Bourgeois, "A Genetic Algorithm-Based Method for Feature Subset Selection," *Soft Comput*, vol. 12, pp. 111-120, 2008.

[20] Stefania Gallova, "Neuro-Fuzzy Learning and Genetic Algorithm Approach With Chaos Theory Principles Applying for Diagnostic Problem Solving," *Proceedings of the World Congress on Engineering, London*, vol. 1, July 1-3, 2009.

[21] Andrei Dinu, Marcian N. Cirstea and Silvia E. Cirstea, "Direct Neural-Network Hardware-Implementation Algorithm," *IEEE Trans. on Industrial Electronics*, vol. 57, no. 5, pp. 1845-1848, May 2010.

[22] Ivan Tsmots and Oleksa Skorokhoda, "Hardware Implementation of the Real Time Neural Network Components," *7th International conference on Perspective Technologies (Memstech 2011)*, pp. 124-126, 11-14 May 2011.

[23] Sue Ellen Haupt, Randy L. Haupt and George S. Young, "A Mixed Integer Genetic Algorithm Used in Biological and Chemical Defense Applications," *Soft Comput*, vol. 15, pp. 51-59, 2011.

[24] P. Karthikeyan, S. Baskar and A. Alphones, "Improved Genetic Algorithm Using Different Genetic Operator Combinations (GOCs) for Multicast Routing in Ad hoc Networks," *Soft Comput (In press)*, 2012.

[25] Adnan Yakzan, Roger Green and Evor Hines, "A Neuro-Genetic Hybrid Algorithm Utilizing Outdoors LOS Optical Wireless Channels," *4th IEEE International conference on Computational Intelligence, Communication Systems and Networks*, pp. 41-44, 2012.

[26] S. C. Nayak, B. B. Misra and H. S. Behera, "Evaluation of Normalization Method on Neuro-Genetic Models for Stock Index Forcasting," *World Congress on Information and Communication Technologies (WICT)*, pp. 602 – 607, 2012.

[27] Asmaa Al-Naqi, Ahmet T. Erdogan and Tughrul Arslan, "Adaptive Three-Dimensional Cellular Genetic Algorithm for Balancing Exploration and Exploitation Processes", *Soft Comput*, vol. 17, pp. 1145-1157, 2013.

[28] A. Arabali, M. Ghofrani, M. Etezadi-Amoli, M. S. Fadali and Y. Baghzouz, "Genetic-Algorithm-Based Optimization Approach for Energy Management," *IEEE Trans. on Power Delivery*, vol. 28, no. 1, pp. 162-170, Jan 2013.

[29] Baher Abdulhai, Himanshu Porwal and Will Recker, "Short-Term Traffic Flow Prediction Using Neuro-Genetic Algorithms," *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*, vol. 7, no. 1, pp. 3-41, 2002.

[30] MATLAB Neural Network Toolbox USA, MATH-WORKS Inc.

[31] UCI Machice Learning Repository, http://archive.ics.uci.edu/ml/datasets/Iris