A Genetic Approach With Guided Mutation For Constructing Neural Network

by

Md. Moinul Hossain and Shuvro Sarker

A THESIS SUBMITTED TO THE DEPT. OF COMPUTER SCIENCE AND ENGINEERING, BUET, IN PARTIAL FULFILMENT OF THE DEGREE OF

BACHELOR OF SCIENCE IN ENGINEERING

Dept. of Computer Science and Engineering

Bangladesh University of Engineering and Technology

February, 2011

Acknowledgements

The authors would like to express their profound gratitude to Dr. Md. Monirul Islam, Professor, Dept. of Computer Science and Engineering, BUET, Dhaka, for his enormous support throughout the time period of this thesis in every conceivable way.

Abstract

The architecture of a neural network plays a vital role in the learning process. This thesis introduces a genetic approach with guided mutation to determine the optimal architecture of a neural network. This approach uses two constructive algorithms, dynamic node creation algorithm(DNC) and a modified version of marchand's algorithm as a part of its mutation process. A technique has been devised in this thesis to train a neural network with the modified marchand's algorithm which is partially trained by DNC and vice versa. This technique is applied in the mutation process of the algorithm proposed. The experimental results shows that, this approach can find a compact architecture for a two class problem with minimal number of hidden neurons and near optimal error rate on test data.

Contents

List of Figures

List of Tables

List of Algorithms

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

The artificial neural networks(ANNs) offer an attractive paradigm for the design and the analysis of adaptive intelligent systems for a wide range of applications in artificial intelligence [10, 11]. The performance of a neural network greatly depends on the type of the network (*e.g.*, feedforward, recurrent), the learning process used and the architecture of the network. An optimal architecture for a network is an important factor of performance. The optimal architecture is a network large enough to learn the underlying function, and as small as possible to generalize well [2]. If an architecture is smaller than the optimal, it cannot learn the underlying function properly because of its limited information process capability. On the other hand if the architecture is larger than the optimal, it overfits the training data of the function to be trained, because of its excess information processing capability. Both overfitting and underfitting causes bad generalization and undesired aspect of using ANNs [15]. So, it is important to find the optimal architecture of a neural network to solve different problems efficiently.

In past several years different algorithms has been proposed to determine the optimal architecture of a neural network. The algorithms can be divided in several types *e.g.*, constructive, pruning, constructive-pruning, regularization etc. A constructive algorithm starts with a minimal architecture and adds hidden layers, neurons, and connections incrementally. A pruning algorithm does the opposite, *i.e.*, it starts with a random network larger than necessary and removes unnecessary parts of the network step by step. A constructive-pruning algorithm is a hybrid approach that invokes both constructive and pruning approach. Constructive approaches are generally preferred over the pruning approaches for several reasons [26]. In addition, evolutionary approaches, such as genetic algorithms have also been used for determining optimal architecture of ANNs automatically. The detailed description for determining optimal architecture of ANNs using constructive, pruning, constructive-pruning and evolutionary approaches are presented in chapter 2.

1.2 OBJECTIVE OF THIS WORK

In this thesis, a new genetic approach is devised for determining the optimal architecture of ANNs. This approach uses two existing constructive algorithms in its mutation process. One of these algorithm is dynamic node creation algorithm(DNC) and the other is a modified version of the marchand's algorithm which is proposed by us in this thesis. In the genetic approach devisd here, the individuals of the initial population are some partially trained networks produced by DNC and the modified marchand's algorithm by running the algorithms for a given amount of time. In case of DNC several networks are formed by varying the parameters of backpropagation(*i.e.*, learning rate,momentum factor etc.). In case of modified marchand's algorithm several networks are formed by the training with different permutations of input vector. A function based on number of hidden neurons and validation error rate is used as the selection function of the genetic algorithm. In each generation of the genetic algorithm a network is selected among the partially trained networks according to the selection function. The best network after each generation is then mutated to produce further individuals for the next generation. The mutation process is guided by the two above mentioned constructive algorithms. These two algorithms are applied again for a certain amount of time to that partially trained network independently to build several extended instances of the network by varying different parameters of the two algorithms and the process repeats until a termination criterion is met. The challenge encountered in this phase of the genetic algorithm was to train a partially trained network by modified marchand's algorithm which was trained by DNC partially in previous generation and train a partially trained network by DNC which was trained by modified marchand's algorithm partially in previous generation. A technique has been devised in this thesis to do this job. As a result of this technique, the networks created in every generation are trained in a "hybrid" way. The algorithm stops when its validation error increases for a certain number of time monotonously.

1.3 THESIS ORGANIZATION

The rest of the chapters in this thesis is organized as follows:

In chapter 2, different existing methods to determine the optimal architecture are discussed. Idea of pruning algorithm, details of some existing constructive algorithms (*e.g.*, upstart algorithm, tilling algorithm etc.) and an overview on genetic approach are also discussed in chapter 2.

Chapter 3, shows the algorithm devised in this thesis including the modification proposed over the marchand's algorithm. The motivation and detailed description of the

algorithm is presented there.
Chapter 4 shows the experimental results and comparison of the genetic approach with DNC and modified marchand's algorithm.
Chapter 5 includes conclusion and ending remarks on future work.

Chapter 2

METHODS OF ARCHITECTURE OPTIMIZATION

2.1 METHODS OF ARCHITECTURE OPTIMIZATION

There are many different approaches proposed over past several years for architecture optimization of ANNs. These approaches can be divided into categories based on their principle and goals:

- Brute force method
- Pruning algorithms
- Regularization technique
- Constructive algorithms
- Constructive-pruning algorithms
- Evolutionary approaches

A brute force method can be thought as the simplest approach to find an optimal architecture. It is based on successive training of networks from smaller to larger, until the smallest topology is found, which still fits the data. The process is very time consuming, because many networks have to be trained. Besides it does not ensure an optimal architecture. Some of the methods incrementally increases the number of neurons starting with a minimal architecture (*e.g.* Constructive algorithms), while others decreases the number of neurons starting with a larger architecture(*e.g.* Pruning algorithms). Some algorithms uses both the techniques (*i.e.*, constructive-pruning algorithms). A review on constructive approaches and pruning approaches can be found in [26] and [22] respectively. It is found that constructive algorithms are more promising then the pruning algorithms [26]. Different algorithm based on evolutionary approaches has also been devised [24, 25]. It is known that a three layered ANN can solve any linear or non linear problem [12–14, 21]. In this thesis the main concentration is given on two constructive algorithms (DNC and modified marchand) which produces three layered feedforward neural network. Details of some constructive algorithms (including DNC and marchand's algorithm) is discussed in this chapter. An overview on genetic approach is discussed further. Before going in to details of constructive and genetic approaches we review the other approaches in short.

2.2 PRUNING ALGORITHMS

The principle of pruning algorithms [5, 16, 20] is to train a network larger than necessary and then remove redundant parts of the network step by step until a optimal architecture is found. The outline of training and pruning procedures is shown in algorithm 2.1.

Algorithm 2.1 principle of Pruning algorithm

```
1: Train a BP network with a reasonable architecture
2: while (the error is higher than a given threshold) do
3:     Compute the relevance of hidden neurons or weights using the chosen heuristic
4:     Remove the least relevant element
5:     Retrain the network
6: end while
```

The pruning approach has several shortcomings [26]:

- Its not possible to determine the initial size of the network to be trained.
- This method is computationally expensive as the majority of the training time is spent to train networks larger than necessary.
- Many network with different size may be capable of implementing acceptable solution. So the pruning approach may not be able to find the smallest acceptable architecture as it starts with a large network.
- The pruning algorithms generally measure the change in error when a hidden unit or weight in the network is removed. Such changes may introduce large errors. Regularization [1, 4, 9] technique solves some of these problems but it requires a balance between the error term and the penalty term introduced in regularization. The penalty term tends to create additional **local minima** in which the algorithm may get stuck while searching for a good solution.

Because of these deficiencies of pruning algorithms the constructive approach are more promising than the pruning algorithms.

2.3 CONSTRUCTIVE ALGORITHMS

The basic idea behind constructive approach is to start with a small network, then add hidden neurons incrementally until a reasonable architecture is achieved. Before going into the details of constructive approaches we will have a look at the issues in constructive approach.

2.3.1 Issues In Constructive Approach

The problem of finding a suitable network architecture can be viewed as a function space search problem with the search space, initial state, goal state and the control strategy defined by the constructive procedure [26].

A search space is a subspace of certain function function space. It is computationally not possible to search the whole function space. A subspace implemented by a constructive procedure is determined by the way the hidden units are generated. The questions arise in generating hidden units are:

- how the net input to a hidden unit will be computed?
- how a new hidden unit will be connected to an existing network?
- what will be a transfer function for a hidden unit?
- how are the new existing network weights will be determined?

A constructive procedure will have two level of search. The first level searches the nearly optimal network architecture and the second level search for optimal set of weights. An architecture found by a constructive approach will be a multi layer perceptron, there might be a single hidden layer or multiple hidden layer and the hidden units might be fully connected or partially connected.

The next issue is what will be initial architecture to start with? Most of the constructive approach starts with direct connections from inputs to outputs. Training is performed on the initial architecture and hidden neurons are added only when performance is not satisfactory, that is the classes are not linearly separable.

The goal state is also an important issue in constructive approaches. A goal state is an acceptable solution to a classification problem. Whether a solution is good or not depends on how we measure the **closeness** between output function and the target function of the problem. There might be more than one goal state *i.e*, there exist many networks that are sufficiently good approximators. It is usually not necessary to find the globally best network and most constructive procedures just give one of these sufficiently good approximators as solution. It is sometimes desirable to examine multiple solutions before selecting the one that explains the data best. This may be done, for example, by repeating the constructive procedure several times using different initial weights in the learning process.

The **generate and test** methodology is taken in all he constructive procedures, which can be viewed as following pseudocode:

Algorithm 2.2 General strategy for constructive approach

```
1: Generate a possible network according to the control strategy which defines the
   way to search for a solution i.e., the architecture and the weights
2: if (the solution is acceptable) then
3:     exit with current solution
4: else
5:     go back to step 1
6: end if
```

In past years several constructive algorithms with different control strategies has been proposed to optimize the architecture of neural network. In this section we are going to discuss some of these algorithms along with the two algorithms (DNC and Marchand's algorithm) that has been used in this thesis.

2.3.2 Some Constructive Algorithms

Followings are some of the algorithms which are widely used in building neural networks in constructive manner:

- Dynamic Node Creation
- Marchand's Algorithm

- Tiling Algorithm
- M-Tiling Algorithm
- Upstart Algorithm

2.3.2.1 Dynamic Node Creation Algorithm

Dynamic node creation algorithm (DNC) [3] is probably the first-ever constructive algorithm for optimizing archiecture of ANNs. It is a very straightforward approach. It uses backpropagation algorithm [23] as the training method. The algorithm starts with direct connection from input to output. Initially the connection weights are randomly assigned. It adds a new hidden neuron and starts training. After some iteration if the error is not minimized then it adds a new node in the hidden layer. This procedure is continued until the error is minimized to a threshold level. After execution of this algorithm a three layered network with a single hidden layer is found. A pseudocde for DNC is shown in algorithm 2.3.

Algorithm 2.3 Dynamic Node Creation

```
1: Create an initial network with connection from input to output. Initialize the
   weights randomly
2: Train the network by backpropagation algorithm for a certain number of epochs
3: Calculate the error of the network. Stop the training process if the threshold limit
   is reached.Otherwise continue
4: Add a new neuron in hidden layer if the error s not significantly minimized after
   some epochs.
5: Go to step 2
```

2.3.2.2 Marchand's Algorithm

This algorithm [18] was proposed by M. Marchand, M. Golea and P. Rujan in 1990. This algorithm produces a feedforward network with one hidden layer. Each node of the network computes a step function with outputs $\in \{0, 1\}$. Hidden nodes are successively added to the network, such that the $(k+1)th$ hidden node performs some non redundant classification over and above what has been performed by the first k hidden nodes. A pseudocode is provided in algorithm 2.4 to describe the algorithm. In the description of the algorithm, T_k^+ and T_k^- represent the training samples of two classes that remain to be correctly classified at the *kth* iteration. At each iteration, a node's weights are trained such that either $|T^+_{k+1}| < |T^+_k|$ or $|T^-_{k+1}| < |T^-_k|$, and $T^+_{k+1} \cup T^-_{k+1} \subset T^+_k \cup T^-_k$, ensuring that the algorithm terminates eventually at the *mth* step, when either T^-_m or T^+_m is empty.

Algorithm 2.4 Marchand's Algorithm

```
1:  $k \leftarrow 0$
2:  $T^+_0 \leftarrow$ set of training samples of class 1
3:  $T^-_0 \leftarrow$ set of training samples of class 2
4:  while ($T^+_k \neq \phi$) and ($T^-_k \neq \phi$) do
5:      $k \leftarrow k + 1$
    Execute any of the following cases:
6:      Case 1:
7:          Obtain a new ($kth$) hidden unit such that its output is 0 for all patterns
    currently $\in T^+_k$ and its output is 1 for at least one of the remaining patterns $\in T^-_k$
8:          Add this unit to the set of nodes $H^-$
9:          $T^+_{k+1} \leftarrow T^+_k$
10:         $T^-_{k+1} \leftarrow$ all samples $\in T^-_k$ for which the new unit generates an output of 0
11:     Case 2:
12:         Obtain a new ($kth$) hidden unit such that its output is 0 for all patterns
    currently $\in T^-_k$ and its output is 1 for at least one of the remaining patterns $\in T^+_k$
13:         Add this unit to the set of nodes $H^-+$
14:         $T^-_{k+1} \leftarrow T^-_k$
15:         $T^+_{k+1} \leftarrow$ all samples $\in T^+_k$ for which the new unit generates an output of 0
16: end while
```

Eventually, when this algorithm has been executed until termination, we have two sets of hidden units (H^+ and H^-), that together give the correct answers on all samples of each class. A new output unit is chosen, with a connection from each of the nodes in $H^+ \cup H^-$, such that the connection from the *kth* hidden node to the output node carries the weight w_k defined as follows:

$$w_k = \begin{cases} 1/2^k & \text{if the } kth \text{ node belongs to } H^+ \\ -1/2^k & \text{if the } kth \text{ node belongs to } H^- \end{cases}$$

The above weight assignment scheme ensures that higher weight is given to the earliest generated nodes. If an input sample was correctly classified by the *jth* node and not by any of the preceding nodes, then

- the contribution to the weighted sum of hidden node outputs is 0 from hidden nodes numbered 1 to $j-1$;
- the contribution to the net from the hidden node numbered j is $1/2^j$ assuming the sample belongs to T_0^+, and $-1/2^j$ otherwise; and
- the contribution to the net from the hidden nodes numbered $> j$ is $\sum_k \pm 1/2^{j+k}$, whose magnitude is less than $1/2^j$.

Hence the net weighted input to the output node is either positive or negative, depending on the class membership of the sample.

2.3.2.3 Tiling Algorithm

The tiling algorithm [19], proposed by Mezard and Nadal (1989), creates a strictly feedforward neural network in which connections exist from each layer only to the immediately succeeding layer. Within each new layer, nodes are successively added until every pair of samples of different classes can be differentiated by some node in that layer. Each layer is constructed in such a way that if two samples i_1 and i_2 belong to different classes, then some node in the layer produces different outputs for i_1 and i_2. The representations of each such layer are then considered to be "faithful" to the desired task; if this was not the case, subsequent layers of nodes would be incapable of distinguishing between samples of different classes i.e., there will be two samples of different classes for which the network outputs will be the same.

Each subsequent layer will have fewer nodes than the preceding layer, until there is only one node in the outermost layer for the two-class classification problem. This ensures termination.

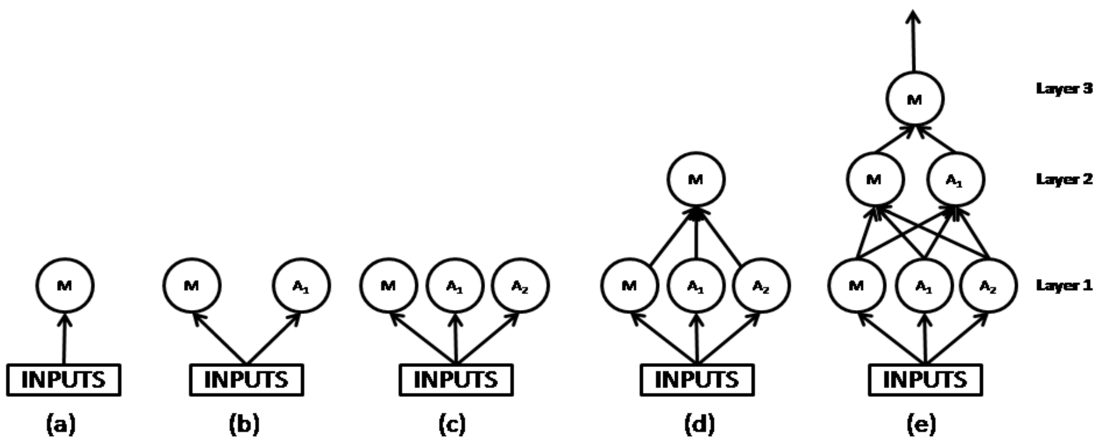

Figure 2.1: Development of a network in tiling algorithm - In fig (a) a master neuron is added. Additional ancillary units are added until the layer 1 is faithful(fig(b) and (c)). Further layers are added in the same manner until the classification task is done(fig (d) and (e)).

Algorithm 2.5 Tiling Algorithm

```
1: Set $L \leftarrow 2$                    ▷ L indicates the current layer. Layer 1 is input layer
2: Add a master neuron M at layer L and train it using pocket algorithm
3: if (M can correctly classify all the training examples) then
4:     Exit with current architecture
5: else
6:     while (layer L is not faithful) do ▷ Ancillary cells are added until the network becomes faithful
7:         Find a subset of training examples of maximal size with more than one classification that produces the same activation for all cells in layer L
8:         Using pocket algorithm train a new ancillary neuron using only the subset of training examples from step 7
9:     end while
10: end if
11: Set $L \leftarrow L + 1$ and go to step 2                    ▷ Layer L is now faithful
```

The process of adding nodes in a given layer is called **tiling**. First, a **master** unit is trained, using the pocket algorithm [8], to perform the best possible dichotomization on its inputs from the preceding layer. Then, additional **ancillary** units are added one after another, attempting to separate the remaining samples of different classes that could not be separated by the units added so far to that layer. Network development using the tiling algorithm is illustrated in figure 2.1.

2.3.2.4 M-Tiling Algorithm

The name M-tiling stands for ”multi category tiling”. This algorithm is an extended version of the tilling algorithm [19]. This algorithm can be applied for a problem with multiple output class where the tiling algorithm works for only problems of two output classes.

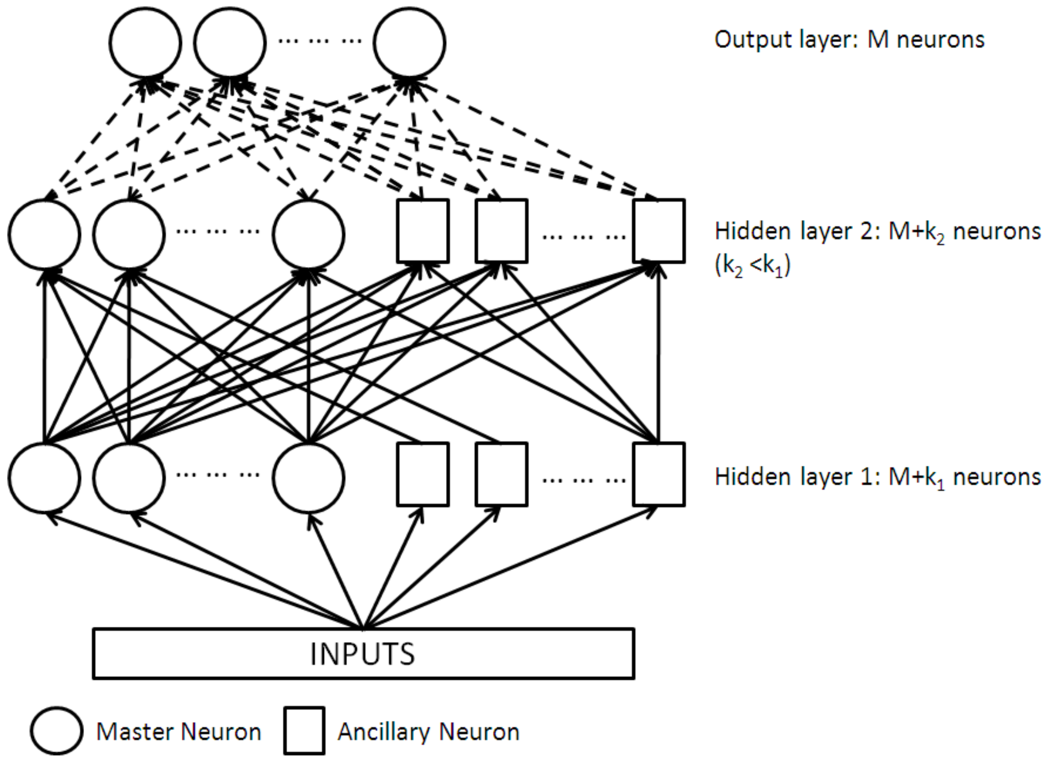
Output layer: M neurons
Hidden layer 2: M+k_2 neurons (k_2 <k_1)
Hidden layer 1: M+k_1 neurons
INPUTS
Master Neuron
Ancillary Neuron

Figure 2.2: The M- tiling algorithm - A typical tiling network developed according to this algorithm.

According to this algorithm every new layer is constructed with M **master** neurons for M output classes. Sets of one or more **ancillary** units are trained at a time in an attempt to make the current layer **faithful**. Figure 2.2 shows the construction of a tiling network.

Algorithm 2.6 M-tiling Algorithm

```
1: Train a layer of $M$ **master** neurons. Each **master** neuron is connected to the $N$ inputs
2: while (The master neurons $M$ of the current layer cannot achieve the desired classification) do
3:     if (The current layer is not faithful) then
4:         Among all the unfaithful output vectors at the current output layer identify the one that the largest number of input patterns map to.(An output vector is said to be unfaithful if it is generated by input patterns belonging to different classes.)
5:         Determine the set of patterns that generate the output vector identified in step 4. This set of patterns will form the training set for ancillary neurons.
6:         Add a set of $1 \leqslant k \leqslant M$ ancillary units where $k$ is the number of target classes represented in the set of patterns identified in the step 5 and train them.
7:     else
8:         Train a new layer of $M$ master neurons that are connected to each neuron in the previous layer.
9:     end if
10: end while
```

2.3.2.5 Upstart Algorithm

The Upstart Algorithm [7] is one of the important learning algorithms that builds a neural network in a constructive manner. This algorithm results a multilayer hierarchical network for a binary classification problem that is guaranteed to converge. The training method used to train each node is the Pocket Algorithm [8]. Frean's algorithm build the network structure by adding hidden units as they are needed in a recursive manner. The network structure found in the upstart algorithm is a binary tree like structure.

- **Wrongly ON**: The target output was 0 but the neuron gives an output of 1.
- **Wrongly OFF**: The target output was 1 but the neuron gives an output of 0.

The basic idea behind the recursive routine is that a hidden units builds other units to correct its mistakes. Initially a hidden unit (let α) is trained by pocket algorithm. This unit α can make two kind of mistakes over input patterns:

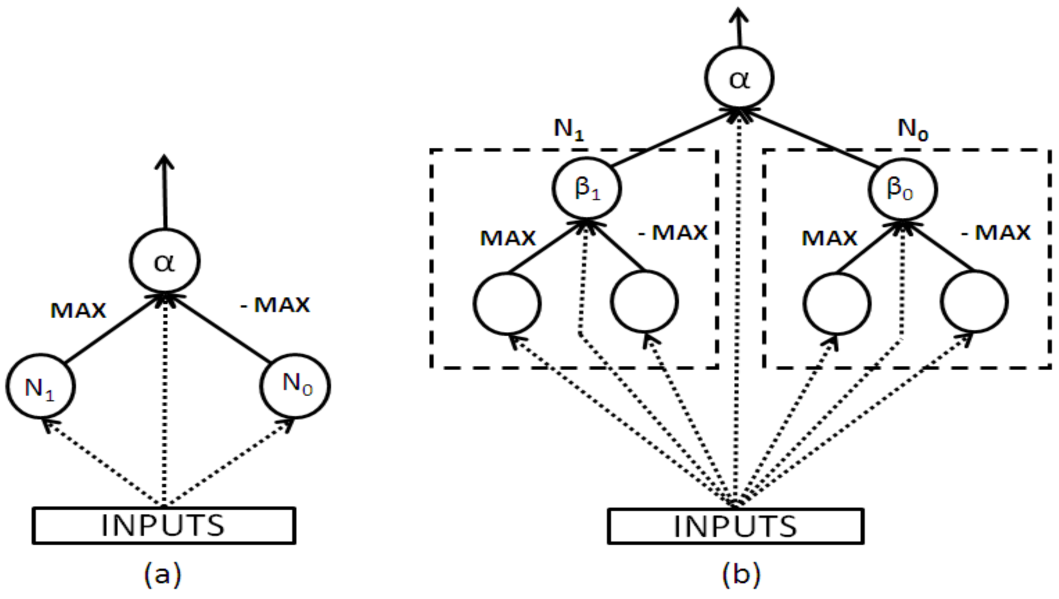

Figure 2.3: The network development in Upstart algorithm - In fig (a) hidden neuron α is added and N_0 and N_1 units are added to correct the misclassification made by α. In fig (b) recursively new neurons are added to N_0 and N_1 to correct the misclassification of N_0 and N_1 respectively.

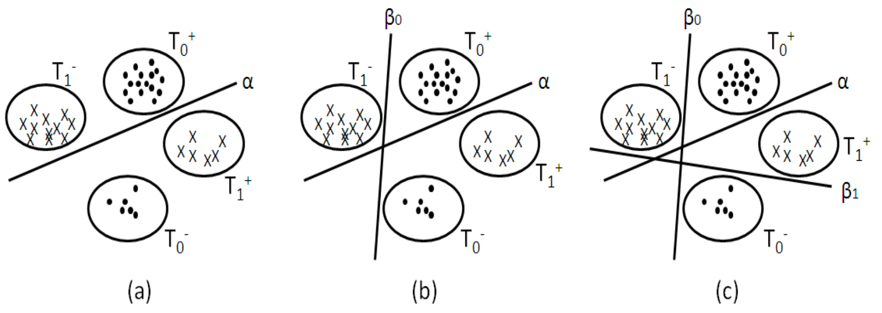

Figure 2.4: Claasification in upstart algorithm - Tasks accomplished by each node in the upper layers of the network developed using the Upstart algorithm. Crosses indicate samples with desired output 1, and circles indicate samples with desired output - 1.

Now, if we consider the patterns for which α is wrongly ON, the output of α can be corrected by a large negative weight from a new unit(let N_0) which is active only for those patterns. Similarly when α is wrongly OFF, the output can be corrected by a large positive weight from another unit (let N_1). Hence N_0 and N_1 can be trained with targets which depend on response of α. These new units(*i.e.*, N_0 and N_1) are called "daughters" as they are generated by the "parent" unit α. Recursively the algorithm is called for every daughter node until no misclassified training pattern is remained. This algorithm is guaranteed to terminate because every recursive call to the Upstart algorithm is invoked with a smaller number of training samples (see fig 2.4). A formal pseudocode for the algorithm is provided in algorithm 2.7.

Algorithm 2.7 Upstart Algorithm

```
1: procedure UPSTART($T_1, T_0$)
2:     $T_1$ is the set of training samples with desired output 1.
3:     $T_0$ is the set of training samples with desired output -1.
4:     Train a perceptron(a node) $\alpha$ using the pocket algorithm to obtain a set of
   weights that perform as well as possible on $T_1 \cup T_0$
5:     while $T_1 = \phi$ OR $T_0 = \phi$ do
6:         if Any samples of $T_1 \cup T_0$ are misclassified by $\alpha$ then
7:             Let $T_1^+ \subseteq T_1$ and $T_0^+ \subseteq T_0$ be the samples correctly classified by $\alpha$ and
   $T_1^- \subseteq T_1$ and $T_0^- \subseteq T_0$ be the samples misclassified by $\alpha$
8:             $N_1 \leftarrow$Upstart$(T_1^-, T_0)$
9:             $N_0 \leftarrow$Upstart$(T_0^-, T_1)$
10:            $MAX = \max_{p \in T_1 \cup T_0} |\sum_k w_k i_k^p|$
11:            (Here, $w_k$ is the weight connecting $k^{th}$ input to $\alpha$ and $i_k^p$ is the $k^{th}$ input
   of pattern $p$)
12:            Add a connection with weight $w_{\alpha,N_1} = MAX$ from the output node of
   $N_1$ to the current node $\alpha$
13:            Add a connection with weight $w_{\alpha,N_0} = -MAX$ from the output node of
   $N_0$ to $\alpha$
14:        end if
15:    end while
16: end procedure
```

2.4 Overview of genetic approach

A genetic algorithm is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

In a genetic algorithm, a population of instances of a problem (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem and evolves toward better solutions. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated.

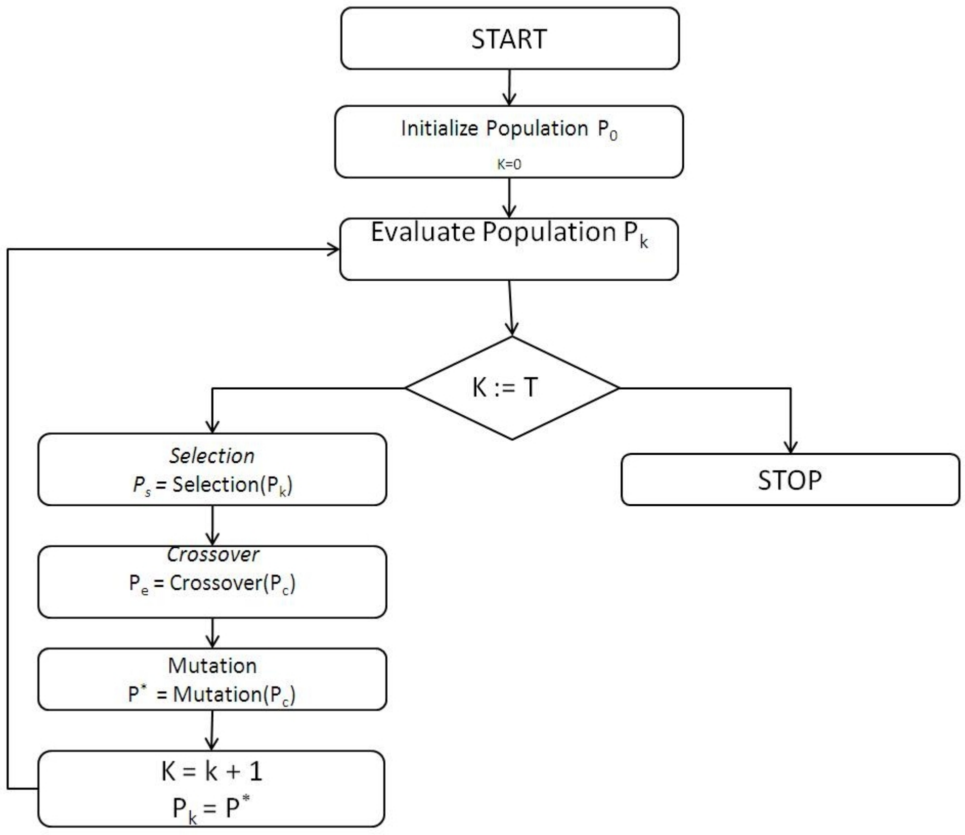

Figure 2.5: - Standard genetic algorithm

Based on the fitness multiple individuals are probabilistically selected from the current population, and modified (with random crossover and mutation) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. A standard approach of this technique is provided in figure 2.5.

The approach devised in this thesis presents a guided mutation for the evolution of the individuals. Crossover is not used as a part of the approach.

Chapter 3

THE GENETIC APPROACH WITH GUIDED MUTATION

3.1 MOTIVATION

The approach devised in this thesis is a mutate only genetic approach. The mutation process is a guided mutation. Two constructive algorithm, dynamic node creation and a modified version of the marchand's algorithm is used in its mutation process.

The two constructive algorithms, DNC and marchand's algorithm has different advantages and disadvantages. The backpropagation learning algorithm used in DNC has several shortcomings, which includes moving target problem, step size problem and problem of determining the learning rate for a specific problem. But it works well on test data when training is done with good tuning of the parameters. On the other hand marchand's algorithm is free from the moving target problem and step size problem as it does not use the back propagation for the training process. But its behaviour on test data may not be always predictable. What we observe is the advantages and disadvantages of this two algorithms are mutually exclusive. Another important observation is in both the algorithm a three layer network is produced. These leads to an intuitive idea that, if we can combine the advantages of these two algorithms a better result can be achieved.So we devised a genetic approach with guided mutation to do the job. Before going into the details of the process let us have a look at the modification done on marchand's algorithm.

3.2 THE MODIFICATION ON MARCHAND'S ALGORITHM

In the marchand's algorithm [18] (see section 2.3.2.2, algorithm 2.4), it tries to classify any one of the two classes (let class 1) at each iteration. It searches for a set of weights randomly that can classify at least one of the instances of class 1. As the algorithm progresses it becomes more and more difficult to find a set of weights to classify the misclassified instances of class 1. It is obvious because with the progress of the algorithm it becomes more and more difficult to find a hyperplane as the search space becomes complex.

Algorithm 3.1 Modified Marchand's Algorithm

```
1: $k \leftarrow 0$
2: $T_0^+ \leftarrow$ set of training samples of class 1
3: $T_0^- \leftarrow$ set of training samples of class 2
4: while ($T_k^+ \neq \phi$) and ($T_k^- \neq \phi$) do
5:     $k \leftarrow k + 1$
6:     $TIMER \leftarrow initialize$
7:     while ($k^{th}$ hidden unit is found such that its output is 0 for all patterns currently $\in T_k^+$ and its output is 1 for at least one of the remaining patterns $\in T_k^-$) and ($TIMER$ is not exceeded) do
8:         Add this unit to the set of nodes $H^-$
9:         $T_{k+1}^+ \leftarrow T_k^+$
10:        $T_{k+1}^- \leftarrow$ all samples $\in T_k^-$ for which the new unit generates an output of 0
11:    end while
12:    if ($TIMER$ is exceeded) and ($k^{th}$ hidden unit is not found) then
13:        $TIMER \leftarrow initialize$
14:        while ($k^{th}$ hidden unit is found such that its output is 0 for all patterns currently $\in T_k^-$ and its output is 1 for at least one of the remaining patterns $\in T_k^+$) and ($TIMER$ is not exceeded) do
15:            Add this unit to the set of nodes $H^-+$
16:            $T_{k+1}^- \leftarrow T_k^-$
17:            $T_{k+1}^+ \leftarrow$ all samples $\in T_k^+$ for which the new unit generates an output of 0
18:        end while
19:    end if
20: end while
```

As a result the convergence rate is little slower.

We have modified the algorithm to make the progress faster. The pseudocode for modified version of the algorithm is shown in algorithm 3.1. The marchand's algorithm tries to classify all the instances of one class. So gradually the complexity of finnding a new hyperplane increases. What this modification does is, if it cannot find a new hyperplane to classify the misclassified instances of a specific class within a certain amount of time it switches to find a hyperplane for the misclassified instances of the other class. This process continues until any of the classes is classified. In this approach the overall complexity in finding a new hyperplane is reduced.

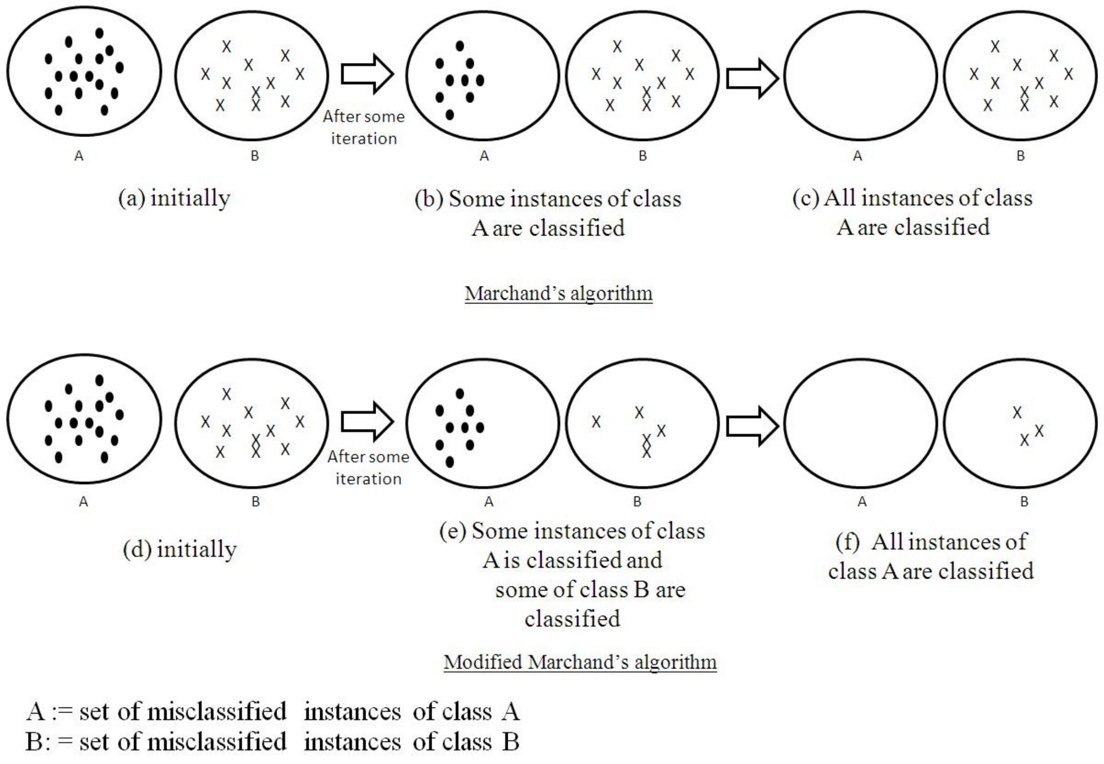
A
B
After some iteration
(a) initially
(b) Some instances of class A are classified
(c) All instances of class A are classified
Marchand's algorithm
(d) initially
(e) Some instances of class A is classified and some of class B are classified
(f) All instances of class A are classified
Modified Marchand's algorithm
A := set of misclassified instances of class A
B: = set of misclassified instances of class B

Figure 3.1: Learning in modified marchand's algorithm - Fig. (a), (b) and (c) shows the condition of the misclassified instances of the class A and B with the progress of the marchand algorithms. Fig. (d), (e) and (f) shows the condition of the misclassified instances of the class A and B with the progress of the modified marchand algorithms.

Simulation result shows a significant change in progress rate with a suitable choice of the $TIMER$ parameter. Figure 3.1 shows the condition of the sets of misclassified instances of both classes with the progress of the algorithms.

3.3 THE GENETIC APPROACH WITH GUIDED MUTATION

3.3.1 Introduction

In our approach the initial population are some partially trained neural networks. Initially some networks (let K networks) are created using dynamic node creation algorithm and marchand's algorithm. In case of dynamic node creation algorithm $k/2$ networks are build which are found by varying learning rate(η), bias value, momentum factor(m) and initial weights from input to output. The networks are trained for a certain amount of time. As a result some partially converged networks are found. On the other hand, in case of marchand's algorithm, some random permutation of the input vector is generated. Training a network for a certain time with different permutation generates different networks which are partially converged. $K/2$ networks are also build in case of marchand's algorithm. These k number of networks are then passed to a selection function.

3.3.2 The Selection Procedure

The selection function gives a fitness score to each of the networks based on the number of their hidden nodes and the mean square error on validation data.

Algorithm 3.2 Genetic Approach with Guided Mutation: Part 1

```
1: procedure SELECT($K$ networks)
2:     $SUM\ OF\ SCORES \leftarrow 0$
3:     for all $K$ networks do
4:         $SCORE \leftarrow \alpha \times X + \beta \times Y$
5:         $SUM\ OF\ SCORES \leftarrow SUM\ OF\ SCORES + SCORE$
6:     end for
7:     for all $K$ networks do
8:         $P \leftarrow 1 - (SCORE/SUM\ OF\ SCORES)$
9:     end for
10:    Select A network $NET$ with probability $P$
11: end procedure
```

The score is measured by the equation:

$$SCORE = \alpha \times X + \beta \times Y$$

Here X denotes the number of hidden neurons and Y denotes the mean square validation error for current network. α and β are user defined coefficients of X and Y respectively where $\alpha + \beta = 1.0$.

Then it associates a probability with each network in such a way that the lower score, the greater is the probability associated with the network. After assigning probability it selects a network with a probability assigned to the network. This selection process is called roulette wheel selection. The selected networks is the next step of the genetic approach which is mutation.

3.3.3 Mutation

This step is a vital step in our algorithm. In this step k number of networks are build trough a guided mutation for the next generation. These K networks are passed again to the selection function. The whole procedure is continued until the termination criterion has met.

In the mutation process $K/2$ networks are build again by applying DNC and $K/2$ networks are build by applying marchand's algorithm on the selected network. But the selected network from the previous generation may be a network which was last constructed by either DNC or marchand's algorithm. So several cases arise when we try to feed a network further by either DNC or marchand's algorithm.

If the network from previous generation was last trained by marchand's algorithm, then its trivial to train it with further DNC or marchand's algorithm. $K/2$ networks are found by just varying the learning rate(η), bias value and momentum factor. But if the network from previous generation was last trained by DNC, then a modification is needed to train the network further with marchand's algorithm. In this case the training data is used on the partially converged network to find the misclassified instances. Tow sets are constructed for misclassified instances of the two classes. Again different permutation of these misclassified instances are generated.

Algorithm 3.2 Genetic Approach with Guided Mutation:: Part 2

```
12: procedure MUTATE($NET$)
13:     There are two possible case when this procedure gets a network as input
        CASE 1: network $NET$ was last trained by DNC
14:         Build $K/2$ networks by applying DNC further on the network $NET$ by
            varying learning rate($\eta$), bias value and momentum factor
15:         Build $K/2$ networks by applying Marchand's algorithm on the network $NET$
            in the following way
            {
16:         Calculate the output of the network $NET$ by all the training data
17:         Build two sets of misclassified instances from above calculation
18:         Add nodes in the hidden layer for a certain amount of time, keeping the
            current weights of $NET$ unchanged
            }
        CASE 2: network $NET$ was last trained by Marchand's algorithm
19:         Build $K/2$ networks by applying DNC further on the network $NET$ by
            varying learning rate($\eta$), bias value and momentum factor
20:         Build $K/2$ networks by applying Marchand's algorithm for different permu-
            tation of misclassified instances further on the network $NET$ for a certain amount
            of time
21: end procedure
```

Algorithm 3.4 Genetic Approach with Guided Mutation: Part 3

```
22: procedure MAIN($K$,$ERROR\ LIMIT$)    ▷ $K$ is user defined
23:     Create $K/2$ instances of DNC network by varying learning rate($\eta$), bias value, momentum factor and initial weights.Train the network for a certain amount of time
24:     Create $K/2$ instances of marchand's algorithms by using different permutation of input vectors. Train the network for a certain amount of time
25:     $NET \leftarrow$ SELECT($K$ networks)    ▷ Select procedure returns a network
26:     while ($ERROR > ERROR\ LIMIT$) do
27:         $K \leftarrow$ MUTATE($NET$)    ▷ procedure MUTATE produces $K$ networks by mutating the network $NET$
28:         $NET \leftarrow$ Select($K$ networks)
29:     end while
         return $NET$
30: end procedure
```

New nodes are then added in the hidden layer to classify these misclassified instances keeping the current weights unchanged. $K/2$ networks are easily found by training with different permutations of the instances.

3.3.4 Termination Criterion

The training error of an ANN may reduce as its training process progresses. However, at some point, usually in the later stages of training, the ANN may start to take advantage of idiosyncrasies in the training data. It sometimes may overfit the data. One common approach to avoid overfitting is to estimate the validation error during training and stop the training process using a criterion based on the validation error. This algorithm uses a very simple criterion. It terminates when the validation error monotonously increases in n successive steps. The validation error is measured as the mean square deviation from the desired output.

Chapter 4

SIMULATION RESULTS

☐ DATASET DESCRIPTION

☐ PARAMETER SETTINGS

☐ EXPERIMENTAL RESULTS

☐ COMPARISON

4.1 DATASET DESCRIPTION

This section introduces with datasets of some well known benchmark classification problems. The datasets used in this thesis are breast cancer [17], Australian credit card approval [6], heart [6] and Pima Indians diabetes [6] datasets. This datasets are used to measure the performance of the algorithm proposed in this thesis. the data sets of different problems were partitioned into three sets: a training set, validation set, and testing set. The number of examples in these sets are shown in the Table 4.1. For each dataset, 30% of the total data is separated for testing and 20% of the remaining data is separated for validation. The training set was used to train and modify ANN architectures. The validation set was used for training process of ANNs, while the testing set for measuring their generalization ability. A random portioning method was chosen for partitioning the data into these three sets.

Table 4.1: Charecteristics of The Datasets

Problem	Number of				
	input attributes	output classes	training examples	validation examples	testing examples
Credit card	14	2	415	103	171
Breast cancer	9	2	382	95	206
Heart	13	2	150	40	80
Pima diabetes	8	2	384	192	192

4.2 PARAMETER SETTINGS

For all experiments presented here, a bias neuron with a random value between $+1$ and -1 was connected to all neurons of hidden layer. For all networks trained with DNC in the mutation process of the genetic approach devised here, the value of learning rate(η) and momentum factor(m) for DNC was chosen as random between 0.02 to 0.11 and 0.1 to 0.7 respectively.

The Details of the system in which the simulation task is conducted is given in tabular format below:

Table 4.2: System Configuration for Simulation Process

Processor	2.40 GHz
Memory	4GB
Operating System	MAC
Programming language	C++

4.3 EXPERIMENTAL RESULT

Table 4.3 shows the performance of the genetic approach with guided mutation over 20 independent runs. The testing error rate (TER) refers to the percentage of wrong classifications produced by ANNs on the testing set. This experimental result is an evidence of performance of this algorithm. For example, for the card problem, the average TER of ANNs according to this algorithm was 15.0 which is a moderate one. TER was only 4.60 for the breast cancer data. A moderate TER is found for the Heart data.

Table 4.3: Performance of the Genetic Approach with Guided Mutation. All results were averaged over 20 independent runs.

Problem	number of hidden neurons	TER (%)
Credit card	8.25	15
Breast cancer	15.12	4.6
Heart	6	14.62
Pima diabetes	11.21	17

4.3.1 Effect Of Different Parameter Values

As seen in chapter 3, section 3.3 the genetic approach with guided mutation has three parameters. These three parameters are α, β and $TIMER$. Table 4.3 represents the result of the genetic approach with guided mutation for a fixed value of these three parameters. The performance of the algorithm on the four benchmark datasets for different values of these three parameters is sown in Table 4.4. Figure 4.1 and 4.2 shows the ANN design process for Australian credit card approval dataset and heart dataset respectively. The value of α(the weight on number of hidden neurons in selection function) and β(the weight on validation error in selection function) is chosen between 0.10 and 1.0 such that $\alpha + \beta = 1$.

Table 4.4: Performance of Genetic Approach with guided mutation for its different parameter values on Four Benchmark Classification Problems. All Results were Averaged over 20 Independent Runs

Problem	Parameters			number of hidden neurons	TER (%)
	α	β	$TIMER$(sec)		
Diabetes	0.7	0.3	3.0	11.53	17.42
	0.2	0.8	3.0	15.67	17.10
	0.2	0.8	2.0	11.25	16.93
	0.5	0.5	2.0	17.60	20.29
Breast cancer	0.7	0.3	3.0	11.74	24.29
	0.2	0.8	2.0	9.87	17
	0.4	0.6	4.0	13	25.31
	0.9	0.1	2.0	8.11	22.01
Credit card	0.7	0.3	2.0	10.12	28.41
	0.7	0.3	3.0	10.5	27.09
	0.2	0.8	2.0	6.24	23.57
	0.5	0.5	2.0	16.23	33.17
Heart	0.7	0.3	3.0	1.86	23.15
	0.2	0.8	3.0	16.50	7.03
	0.2	0.8	2.0	1.9	16.75
	0.5	0.5	2.0	2.17	21.13

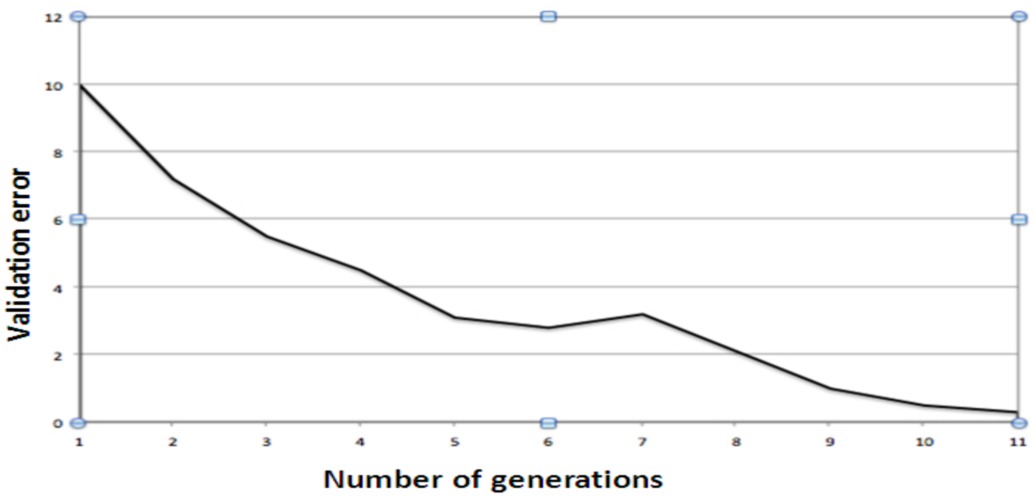

Figure 4.1: ANN design process for Australian credit card approval dataset -

The value of $TIMER$ is a very important parameter. It determines in each generation how much time will DNC and marchand's algorithm will get in each generation to train a partially trained network from previous generation. It can be observed that, when much weight is given on β the training error rate(TER) improves and number of hidden neuron is increased. As for Heart data when $\alpha = 0.2$ and $\beta = 0.8$ the number of hidden neuron becomes 16.50 where for a balanced values of α and β (0.5 and 0.5) it is only 2.17. In case of TER it is reduced to 7.03% whereas for $\alpha = 0.5$ and $\beta = 0.5$ it is 21.13. The important observation is this algorithm can achieve both the goal of minimizing number of hidden neurons and the error rate. So, this algorithm can perform well in the cases where accuracy is the major (by tuning β to a higher value then α) concern as well as in cases where processing overhead is major concern(by tuning α to a higher value then β)

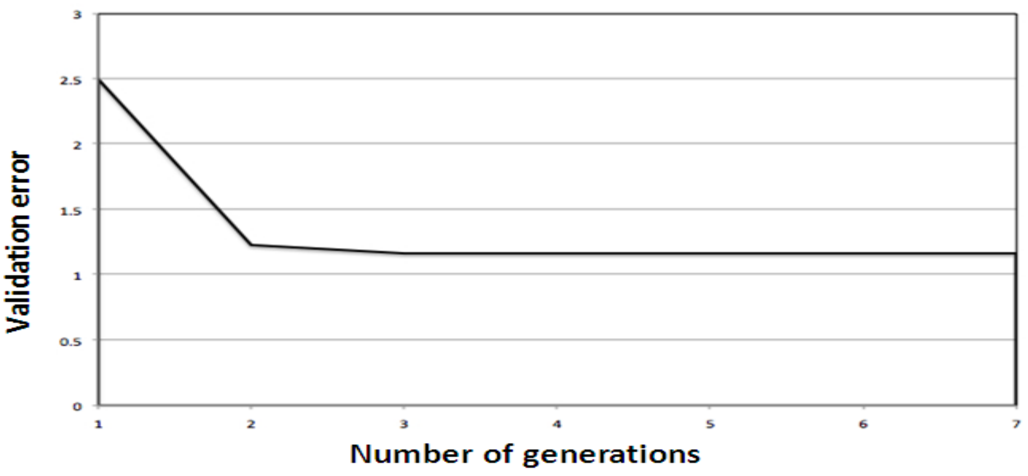

Figure 4.2: ANN design process for heart dataset -

4.3.2 Comparison

Table 4.5 shows a comparison between the genetic approach with guided mutation, DNC and the modfied marchand's algorithm based on the number of hidden neurons.

Table 4.6 shows a comparison between the genetic approach with guided mutation, DNC and the modfied marchand's algorithm based on the testing error rate (TER).

Table 4.5: Comparison of Three algorithms, Genetic Approach with Guided Mutation(GAGM), DNC and MArchand's algorithm Based on Number of Hidden Neurons. All results was Averaged over 20 Independent Runs.

Algorithm	Credit card	Breast cancer	Heart	Pima diabetes
GAGM	8.25	15.12	6.0	11.21
DNC	12.89	14.10	17.10	15.30
Modified Marchand's algorithm	42.35	53.25	3.10	38.64

Table 4.6: Comparison of Three algorithms, Genetic Approach with Guided Mutation(GAGM), DNC and MArchand's algorithm Based on testing Error Rate (TER) Neurons. All results was Averaged over 20 Independent Runs.

Algorithm	Credit card	Breast cancer	Heart	Pima diabetes
GAGM	15	4.6	14.62	17.0
DNC	29.39	3.65	29.40	27.36
Modified Marchand's algorithm	27.21	2.11	21.91	24.74

Chapter 5

Conclusion

The simulation result obtained in our proposed approach is promising. The selection function used in the genetic approach is built based on the number of hidden nodes and the validation error rate. So, this algorithm can be tuned to any of the desired purpose of error accuracy or processing overhead. The algorithm can perform well in the cases where accuracy is the major concern as well as in cases where processing overhead is major concern. For further improvement of the performance, more constructive algorithms may be incorporated as a part of the mutation process. Incorporating pruning algorithms in the mutation process also can be a an option to improve the performance. Other parameters may be incorporated to improve the selection criteria. If the algorithm can be extended for multiple output class then it can be used with a large number of problems.

References

[1] Bernardo A. Huberman Andreas S. Weigend, David E. Rumelhart. Generalization by weight-elimination with application to forecasting. *NIPS*, 3:875–882, 1991. 7

[2] Oya Aran and Ethem Alpaydin. An incremental neural network construction algorithm for training multilayer perceptron, 2003. 2

[3] Timur Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989. 10

[4] Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. In David S. Touretzky, editor, *NIPS*, pages 519–526. Morgan Kaufmann, 1988. 7

[5] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990. 7

[6] A. Frank and A. Asuncion. UCI machine learning repository, 2010. 30

[7] Marcus Frean. The upstart algorithm: a method for constructing and training feed forward neural networks. 1990. 15

[8] Stephen I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions On Neural Networks*, 1(2), 1990. 13, 15

[9] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with backpropagation. Technical Report CSL-Report-36, Cognitive Science Laboratory, Princeton University, 1989. to appear in proceedings of NIPS-88. 7

[10] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991. 2

[11] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1–3):195–234, 1989. 2

[12] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991. 6

[13] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[14] Kurt Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993. 6

[15] Md. Monirul Islam, Md. Faijul Amin, Suman Ahmmed, and Kazuyuki Murase. An adaptive merging and growing algorithm for designing artificial neural networks. In *IJCNN*, pages 2003–2008. IEEE, 2008. 2

[16] E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, June 1990. 7

[17] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming, 2010. 30

[18] M. Marchand, M. Golea, and P. Rujan. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11:487–492, 1990. 10, 22

[19] M Mezard and Jean-P Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of Physics A: Mathematical and General*, 22(12), 1989. 12, 14

[20] M. C. Mozer. Skeletonization: A technique for trimming the fat from a network via relevance assessment. Technical Report CU-CS-421-89, University of Colorado, 1989. 7

[21] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991. 6

[22] Russell Reed. Pruning algorithms — A survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993. 6

[23] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. Backpropagation: the basic theory. In *Backpropagation: Theory, Architectures and Applications*, pages 1–34. Lawrence Erlbaum Associates, Hillsdale, NJ, 1995. 10

[24] J. David Schaffer, Darrell Whitley, and Larry J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Proceedings of COGANN92*, pages 1–37, 1992. 6

[25] Xin Yao. A review of evolutionary artificial neural networks. accepted by *International Journal of Intelligent Systems*, 1993. 6

[26] Tin yau Kwok and Dit yan Yeung. Constructive feedforward neural networks for regression problems: A survey. Technical report, September 11 1995. 2, 6, 7, 8