Chapter 4

The Genetic search scheme

This section introduces different types of evolutionary methodology. Along with the new approach, the effects of the genetic operator crossover upon the MCE encoded ANNs are discussed. The algorithm to realize the PST is also presented.

4.1 Evolutionary Approaches

Evolutionary algorithm (EA) is an umbrella term used to describe computer based problem solving systems which use computational models of evolutionary processes as key elements in their design and implementation. A variety of EAs have been proposed. The major ones are: genetic algorithms, evolutionary programming, evolution strategies and genetic programming. They all share a common conceptual base of simulating the evolution of individual structures via processes of selection, mutation, and reproduction as depicted in Figure 4.1. The processes depend on the perceived performance of the individual structures as defined by an environment. In brief, EA is a system which incorporates aspects of natural selection or survival of the fittest. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

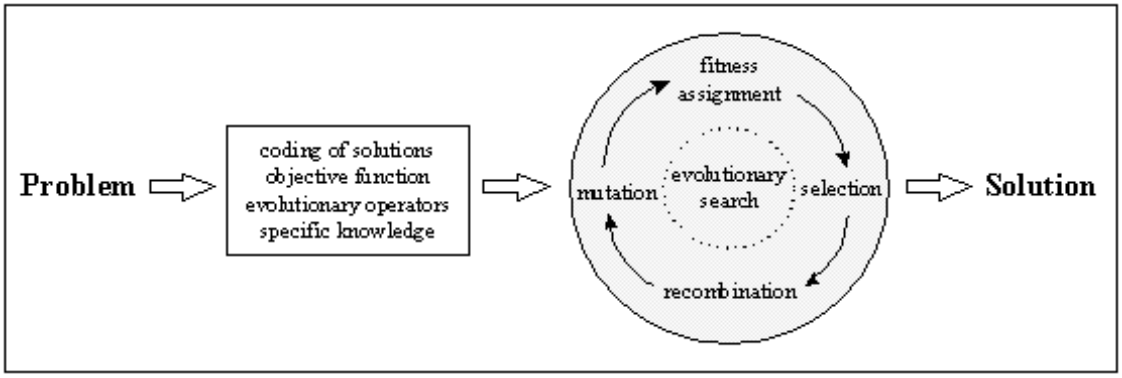

Figure 4.1: Problem solution using EA.

4.1.1 Genetic Algorithm

The genetic algorithm (GA) is a model of machine learning which derives its behavior from a metaphor of the processes of evolution in nature. This is done by the creation within a machine of a population of individuals represented by chromosomes, in essence a set of character strings that are analogous to the base-4 chromosomes that is seen in our own DNA. The individuals in the population then go through a process of evolution.

At the molecular level what occurs is that a pair of chromosomes bump into one another, exchange chunks of genetic information and drift apart. This is the recombination operation, which GA generally refer to as crossover because of the way that genetic material crosses over from one chromosome to another.

The crossover operation happens in an environment where the selection of who gets to mate is a function of the fitness of the individual, i.e. how good the individual is at competing in its environment. Mutation also plays a role in this process, although how important its role is continues to be a matter of debate. When the GA is implemented it is usually done in a manner that involves the following cycle: evaluate the fitness of all of the individuals in the population. Create a new population by performing operations such as crossover, fitness-proportionate reproduction and mutation on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population. One iteration of this loop is referred to as a generation. There is no theoretical reason for this as an implementation model. Indeed, one does not see this

punctuated behavior in populations in nature as a whole, but it is a convenient implementation model.

4.1.2 Evolutionary Programming

Evolutionary programming (EP), originally conceived by Lawrence J. Fogel in 1960, is a stochastic optimization strategy similar to GAs, but instead places emphasis on the behavioral linkage between parents and their offspring, rather than seeking to emulate specific genetic operators as observed in nature. EP is similar to evolution strategies (ES), although the two approaches developed independently. It should be pointed out that EP typically does not use any crossover as a genetic operator [12].

There are two important ways in which EP differs from GA. First, the typical GA approach involves encoding the problem solutions as a string of representative tokens, the genome. In EP, the representation follows from the problem. A neural network can be represented in the same manner as it is implemented, for example, because the mutation operation does not demand a linear encoding. Second, the mutation operation simply changes aspects of the solution according to a statistical distribution which weights minor variations in the behavior of the offspring as highly probable and substantial variations as increasingly unlikely.

The main differences between Evolution strategy (ES) and EP are:

a) Selection: EP typically uses stochastic selection via a tournament. Each trial solution in the population faces competition against a preselected number of opponents and receives a "win" if it is at least as good as its opponent in each encounter. Selection then eliminates those solutions with the least wins. In contrast, ES typically uses deterministic selection in which the worst solutions are purged from the population based directly on their function evaluation.

b) Recombination: EP is an abstraction of evolution at the level of reproductive populations (i.e., species) and thus no recombination mechanisms are typically used because recombination does not occur between species. In contrast, ES is an abstraction

of evolution at the level of individual behavior. When self-adaptive information is incorporated this is purely genetic information (as opposed to phenotypic) and thus some forms of recombination are reasonable and many forms of recombination have been implemented within ES.

4.1.3 Evolution Strategy

Evolution strategies (ES) were invented to solve technical optimization problems. It is more or less similar to EP. Self-adaptation within ES depends on randomness, population size, cooperation and deterioration [21], [40].

4.1.4 Genetic Programming

Genetic programming (GP) is the extension of the genetic model of learning into the space of programs. That is, the objects that constitute the population are not fixed-length character strings that encode possible solutions to the problem at hand, they are programs that, when executed, are the candidate solutions to the problem. These programs are expressed in genetic programming as parse trees, rather than as lines of code.

In GP the crossover operation is implemented by taking randomly selected subtrees in the individuals (selected according to fitness) and exchanging them. It should be pointed out that GP usually does not use any mutation as a genetic operator.

4.2 Crossover on MCE

This subsection illustrates the rules and effects of the genetic operator crossover on the modified CE.

While growing of an ANN from a PST, each cell transformed to a neuron when its reading head ends at an END symbol. This necessarily means that for each neuron there must have an END symbol, which is a leaf of the PST. In fact, the number of ENDs in a

PST is equal to or greater than (some neurons of corresponding END leaves may be removed by MRG operation) the number of neurons in the ANN. To avoid the risk of changing the structure of ANN drastically by crossover effect, crossover is allowed on last level (leaf) or second last level. That is, the roots of the two subtrees chosen form a pair of CE for crossover must have all children as END or no children. If the root of a subtree is a binary symbol (i.e. PAR or SEQ) it has two children both of END symbols. If the root is a unary symbol (i.e. CUT or MRG) it has single child END. If the root is END itself, it has no children. But both of the subtrees chosen for exchange cannot be same since in that case the crossover would have no effect. The following table shows all possible combinations of crossover and their effects on the genotypes of ANNs.

Table 4.1: Effects of crossover on CE.

Effects of Crossover		***Replaced by***				
		PAR	**SEQ**	**CUT**	**MRG**	**END**
To be replaced	**PAR**	NC	OCN	CD, ND	OC, ND	ND
	SEQ	OCN	NC	CD, ND	OC, ND	ND
	CUT	CA, NA	CA, NA	OC or NC	OC, ND*	CA
	MRG	OC, NA	OC, NA	OC, NA*	NC or OCN	OC, NA*
	END	NA	NA	CD	OC, ND*	NC

Here,

NC ≡ no change in the architecture of corresponding ANNs

NA ≡ node addition

ND ≡ node deletion

CA ≡ connection addition

CD ≡ connection deletion

OC ≡ new orientation of connection

OCN ≡ new orientation of both connection and node

* effect may or may not takes place, depends on orientation of connections and the parameter of the corresponding symbol

It is clear from the above table that all sorts of modifications of ANNs (for example addition and / or deletion of node and / or connection) can be happened from crossover. Since crossover is allowed only at the lowest level or immediate above it, massive changes in the architecture at a time cannot occur.

One has to put across the rules for applying the genetic operator crossover on the modified CE so that its properties are not destroyed even after crossover operation. First thing to remember is that crossover cannot be applied on the roots of PSTs, it must be applied on two different subtrees. If these two subtrees are same crossover would have no effect. When applying crossover between two cellular encodings CE1 and CE2 by exchanging two random subtrees ST1 (from CE1) and ST2 (from CE2), the following three situations may arise.

i) None the parents of ST1 and ST2 is PAR.
ii) One of the parents of ST1 and ST2 is PAR.
iii) Both the parents of ST1 and ST2 are PAR.

In case (i), crossover is just applied accordingly.

In case (ii), let, without loss of generality, the parent of ST1 is PAR and the parent of ST2 is not PAR. The sub-cases can occur as follows:

a) Both the roots of ST1, ST2 are PAR
b) Both the roots of ST1, ST2 are not PAR
c) The root of ST1 is PAR, the root of ST2 is not PAR
d) The root of ST1 is not PAR, the root of ST2 is PAR

In sub-case (a) crossover is not applied. For other sub-cases it is applied accordingly. After crossover is done, the ordering described in Property 1 is established among the children of the parent of ST2 in CE1.

In case (iii), the sub-cases can occur as follows:

e) Both the roots of ST1, ST2 are PAR
f) Both the roots of ST1, ST2 are not PAR
g) The root of ST1 is PAR, the root of ST2 is not PAR
h) The root of ST1 is not PAR, the root of ST2 is PAR

In sub-case (e) crossover is not done. For other sub-cases, crossover is applied accordingly but the ordering described in Property 1 is established among the children of both the parent of ST2 in CE1 and the parent of ST1 in CE2.

4.3 CE to DE Conversion

Input: the PST to be executed.
Output: the matrix representation (direct encoding) of the ANN.

$m \equiv$ number of input nodes

$n \equiv$ number of output nodes

$N \equiv$ maximum number of hidden nodes allowed in the ANN

$M \equiv$ a matrix with $m + N$ rows and $N + n$ columns; $M[i, j] = 1$ means that there is a connection from i-th node to j-th node, $M[i, j] = 0$ means that there is no connection from i-th node to j-th node

Initialize a **FIFO** with the root of the PST as the only entry
Initialize all the entries of the matrix M to 0
while (head of the **FIFO** ≠ NULL)
    case (head of the **FIFO**)
        PAR: Copy the inputs and outputs of the current i-th entry (cell) in a new j-th entry (cell) of the matrix. Enter both of the entries (cells) i, j at the end of the FIFO in random order.
        SEQ: Remove the outputs of the current i-th entry and copy these to a new j-th entry of the matrix; Set $M[i, j] = 1$. Enter entry i at the end of the FIFO, then enter entry j in the FIFO.
        CUT *x*: Let the x-th input link to the current entry i is j. Set $M[j, i] = 0$. Place the entry i at the end of the FIFO.
        MRG *x*: Let the x-th input link to the current entry i is j. If j-th entry has only one output link and its reading head not yet end, enter the current

entry *i* at the end of the FIFO. Nothing changes in the matrix (this action is like WAIT). Otherwise, set *M*[*j*, *i*] = 0 and copy all the input links into entry *j* to the entry *i*. Also if *j*-th entry has no more output link, remove all the input links into entry *j*. Place entry *i* at the end of the FIFO.

END: Do nothing. Purge the reading head of the cell and convert it to a node of ANN.

end *case*

end *while*

The number of symbols k executed by the algorithm is the number of symbols present in the PST and $k = O(m + N + n)$. Let, PAR, SEQ and MRG handle on an average l number of connections (entries of the matrix) that depends on $O(m + N)$. So in the worst case, the cost for executing the above algorithm is $O(lk)$.

4.4 The Evolutionary System

The major steps of the evolutionary system proposed in this work are depicted in the Figure 4.2, which are explained further as follows [6], [12], [19], [33]:

1. Generate initial population (program symbol trees) of *M* networks randomly. The initial number of hidden nodes, connection density and weights for each ANN are uniformly distributed at random within certain ranges.

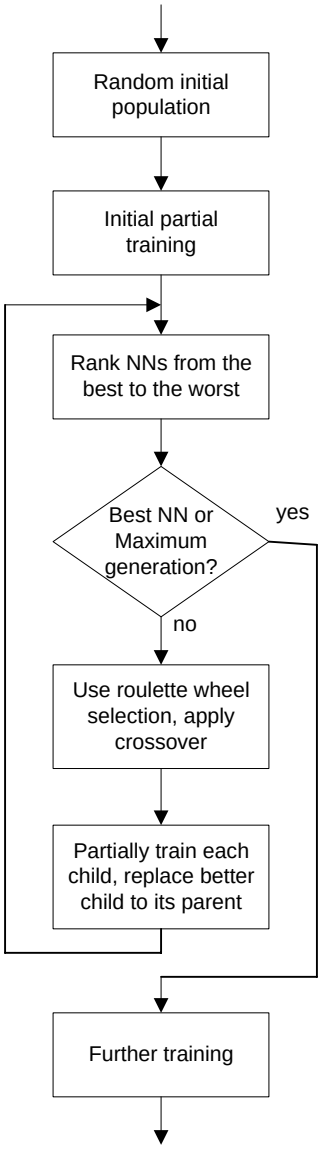

Figure 4.2: Major steps of the evolutionary system.

2. Partially train each network in the population for a certain number of epochs K_0, which is user specified, using normal backpropagation [52].
3. Rank the networks in the population according to their error values, from the best to the worst.
4. If the best network found is acceptable or the maximum number of generations has been reached, stop the evolutionary process and go to the step 7. Otherwise continue.

5. Use the rank based selection (the roulette wheel selection [10]) to select parents for crossover operations to obtain offspring networks.
6. Partially train each child for K_1 epochs using normal BP. Here, K_1 is a user specified parameter. In the evolution if an offspring is better than its parent it will replace the parent. Go to step 3.
7. After the evolutionary process, train the best network further on the combined training and validation set until it converges.

Chapter 5

Experimental Studies

Machine learning investigates the mechanisms by which knowledge is acquired through experience. Databases with millions of records and thousands of fields are now common in business, medicine, engineering, and the sciences. In order to evaluate the ability of the approach in evolving ANN, it has been applied to the database of some real-world problems. This section portrays the experimental references, setup, results and comparisons with other works.

5.1 Data Sets Applied

The algorithm is applied on four real-world problems in the medical domain, i.e., the breast cancer problem, the diabetes problem, the heart disease problem, and the thyroid problem. All data sets were obtained from the machine learning benchmark repository cited at the Department of Information and Computer Science of University of California, Irvine.

This is a repository of databases, domain theories and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. A large collection of data sets is accessible via anonymous FTP at ftp.ics.uci.edu [128.195.1.1] in directory "/pub/machine-learning-databases" or via web browser at http://www.ics.uci.edu/~mlearn/ MLRepository.html. The Knowledge Discovery in Databases (KDD) Archive here encompasses a wide variety of data types, analysis tasks, and application areas. The primary role of this repository is to serve as a benchmark tested to enable researchers in knowledge discovery and data mining to scale existing and

future data analysis algorithms to very large and complex data sets. This archive is supported by the Information and Data Management Program at the National Science Foundation, and is intended to expand the current UCI Machine Learning Database Repository to datasets that are orders of magnitude larger and more complex. An important file to read regarding the repository is the README file. It contains an overall description of the repository. Another file, SUMMARY-TABLE, contains a table of some of the databases. Each database is characterized by a fixed set of attributes. The construction of this repository is an on-going process. The majority of the entries in the repository were contributed by machine learning researchers outside of UCI.

Here the data format followed in Proben1 is used. Proben1 is a collection of 12 learning problems consisting of real data. The data files all share a single simple common format. Along with the data comes a technical report describing a set of rules and conventions for performing and reporting benchmark tests and their results. It is accessible via anonymous FTP on ftp.cs.cmu.edu [128.2.206.173] as/afs/cs/project/connect/bench/ contrib/prechelt/proben1.tar.gz and also on ftp.ira.uka.de as /pub/neuron/proben1.tar.gz. The file is about 1.8 MB and unpacks into about 20 MB.

The four medical diagnosis problems used have the following common characteristics [34].

- The input attributes used are similar to those a human expert would use in order to solve the same problem.
- The outputs represent either the classification of a number of understandable classes or the prediction of a set of understandable quantities.
- In practice, all these problems are solved by human experts.
- Examples are expensive to get. This has the consequence that the training sets are not very large.
- There are missing attribute values in the data sets.

These data sets represent some of the most challenging problems in the ANN and machine learning field. They have a small sample size of noisy data.

5.1.1 Heart Disease

This Heart directory contains 4 databases concerning heart disease diagnosis. The data was collected from the four following locations:

1. Cleveland Clinic Foundation (cleveland.data)
2. Hungarian Institute of Cardiology, Budapest (hungarian.data)
3. V.A. Medical Center, Long Beach, CA (long-beach-va.data)
4. University Hospital, Zurich, Switzerland (switzerland.data)

The first set which comes from the Cleveland Clinic Foundation and was supplied by Robert Detrano of the V.A. Medical Center, Long Beach, CA is used. The purpose of the data set is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. This database contains 13 attributes, which have been extracted from a larger set of 75. The database originally contained 303 examples. There are two classes: presence and absence (of heart disease). This is a reduction of the number of classes in the original data set in which there were four different degrees of heart disease. The input attributes are discrete on a scale 0 - 1 (real) and output is 0 or 1 (binary).

5.1.2 Diabetes

This data set was originally donated by Vincent Sigillito from Johns Hopkins University and was constructed by constrained selection from a larger database held by the National Institute of Diabetes and Digestive and Kidney Diseases. All patients represented in this data set are females of at least 21 years old and of Pima Indian heritage living near Phoenix, AZ. The problem posed here is to predict whether a patient would test positive for diabetes according to World Health Organization criteria given a number of physiological measurements and medical test results. This is a two class problem with class value one interpreted as “tested positive for diabetes.” There are 500 examples of class 1 and 268 of class 2. There are eight attributes for each example. The data set is rather difficult to classify. The so-called “class” value is really a binarised form of another attribute which is itself highly indicative of certain types of diabetes but does not

have a one to one correspondence with the medical condition of being diabetic. Although there are no missing values in this dataset according to its documentation, there are several senseless 0 values. These most probably indicate missing data. Nevertheless, this data are handled as if it was real, thereby introducing some errors (or noise, if you want) into the dataset. The input attributes are discrete on a scale 0 - 1 (real) and output attribute is binary valued.

5.1.3 Thyroid

This data set comes from the “ann” version of the “thyroid disease” data set from the UCI ML repository. Original donor is Randolf Wernero btained from Daimler-Benz.Two files were provided. “anntrain.data” contains 3772 learning examples. “ann-test.data” contains 3428 testing examples. There are 21 (15 attributes are binary, 6 attributes are continuous) attributes for each example. The purpose of the data set is to determine whether a patient referred to the clinic is hypothyroid. Therefore three classes are built: normal (not hypothyroid), hyperfunction and subnormal functioning. Because 92 percent of the patients are not hyperthyroid, a good classifier must be significantly better than 92%. The input attributes are discrete on a scale 0 - 1 (real) and the 3 output attributes (1, 2, or 3) are encoded with a 1-of-3 encoding (1 0 0, 0 1 0, or 0 0 1).

5.1.4 Breast Cancer

The breast cancer data set was originally obtained from Dr. William H. Wolberg (physician) at the University of Wisconsin Hospitals, Madison, Wisconsin, USA. The purpose of the data set is to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. The data set contains nine attributes and 699 examples of which 458 are benign examples and 241 are malignant examples. There are 9 input attributes, all discrete on a scale 0 - 1 (real) and 1 binary output attribute.

5.2 Experimental Setup

All the data sets used have been partitioned into three sets: a training set, a validation set, and a testing set. The training set is used to train ANN by back propagation, the testing set is used to evaluate the performance of the system. The validation set is not used in this work. In the following experiments, according to Proben1, each data set is partitioned as follows.

- For the breast cancer data set, the first 350 examples are used for the training set, the following 175 examples for the validation set, and the final 174 examples for the testing set.
- For the diabetes data set, the first 384 examples are used for the training set, the following 192 examples for the validation set, the final 192 examples for the testing set.
- For the heart disease data set, the first 152 examples are used for the training set, the following 76 examples for the validation set, and the final 75 examples for the testing set.
- For the thyroid data set, the first 3600 examples in “ann-train data” are used for the training set, the next 1800 for the validation set, and the rest 1800 for the testing set.

It, however, should be kept in mind that such partitions do not represent the optimal ones [37]. As said before, the input attributes of the diabetes data set and heart disease data set are rescaled to between 0.0 and 1.0 by a linear function. The output attributes of all the problems are encoded using a 1-of-output representation for classes. The winner-takes-all method is used here, i.e., the output with the highest activation designates the class. There are some control parameters which need to be specified by the user. Most parameters used in the experiments are set to be the same: the population size (20), the learning rate (0.25), initial weight range -0.5 to +0.5 etc. These parameters were chosen after some limited preliminary experiments. They were not meant to be optimal. Actually, enormous amount of experiments are needed for the parameter tuning and the tuned parameter may be sub-optimal since parameters are independent and interacts in very complex ways.

This is true regardless how the parameters are tuned and is based on the observation that a run on an evolutionary algorithm is intrinsically dynamic and adaptive process [1].
In this approach, the selection mechanism used is the elitist roulette wheel scheme, which is described in chapter 2 more details. The error function (inverse of fitness) E is

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

where the O_{max} and O_{min} are the maximum and minimum values of output coefficients in the problem representation, n is the number of output nodes, T is the number of patterns and $Y(i,t)$, $Z(i,t)$ are actual and desired outputs of node for pattern. The values of O_{max} and O_{min} are found from the input data set before the evolution starts. The equation above was suggested by Prechelt [34] to make the error measure less dependent on the size of the validation set and the number of output nodes. Hence a mean squared error percentage is adopted. In this thesis, training error and testing error rate refer two different measures. Training error means the value calculated while training through the error function described above. And testing error rate is the percentage of testing input patterns that are incorrectly classified.

5.3 Results

The program is run for different epochs and different generations. Their ranges vary from one data set to another set. The following table summarizes the results. The Table 5.1 shows average values, standard deviations and best results for number of connections, number of hidden nodes, number of generations and epochs needed, training and testing errors of the ANN evolved. Here the best result (* marked) is for the ANN with the least testing error rate. For the diabetes problem, it has been found in one generation for 200 epochs. So, total time (epoch × generation × ANN) is 200 × 1 × 20 = 4000 only. Average number of epochs needed is 127.96 and average number of generations needed is 9.99. On average an ANN has 6 hidden nodes and 56.7 connections with testing error rate is 0.25108. For thyroid, heart disease, breast cancer problem total time needed is 70000, 13000 and 200 respectively. These are much less than that of the experiments done by others as shown in the next section.

Table 5.1: Experimental outcomes (* ANN with the least testing error rate)

Data Sets		No. of Connections	No. of Hidden nodes	No. of Generations	No. of Epochs	Training Error	Testing Error rate
Diabetes	Mean	56.75555556	6.059259259	9.99259259	127.962	13.001760	0.2510803
	SD	23.90824084	2.278265126	12.30368	141.479	1.7322284	0.0390747
	Best*	**51**	**6**	**1**	**200**	**11.989235**	**0.208333**
Thyroid	Mean	80.0882353	4.441176471	11.0882353	450.588	2.2820206	0.057467265
	SD	40.8429124	2.258870413	15.8733007	883.108	0.9273925	0.031933939
	Best*	**114**	**5**	**1**	**3500**	**0.672626**	**0.027222**
Heart	Mean	204.033058	5.73553719	44.702479	117.809	6.2917089	0.13652889
	SD	100.599696	2.90335609	57.967756	169.049	3.3793083	0.05559175
	Best*	**145**	**4**	**5**	**130**	**4.54072**	**0.053333**
Cancer	Mean	80.3763441	7.77419355	10.2473118	70.5376	2.6638425	0.025806527
	SD	58.6144206	4.72304487	12.7156664	124.189	0.7204406	0.016328747
	Best*	**40**	**4**	**1**	**10**	**3.658594**	**0.005714**

For the diabetes problem the following trend of generation versus error given in Figure 5.1 is found, error is decreased as generation is increased up to certain level. Generation versus connections in Figure 5.2 shows that number of connection is minimized around generation 3. In Figure 5.3, epoch versus error for both generation 5 and 12 are shown. Figure 5.4 indicates, error is minimized when time is near 30000.

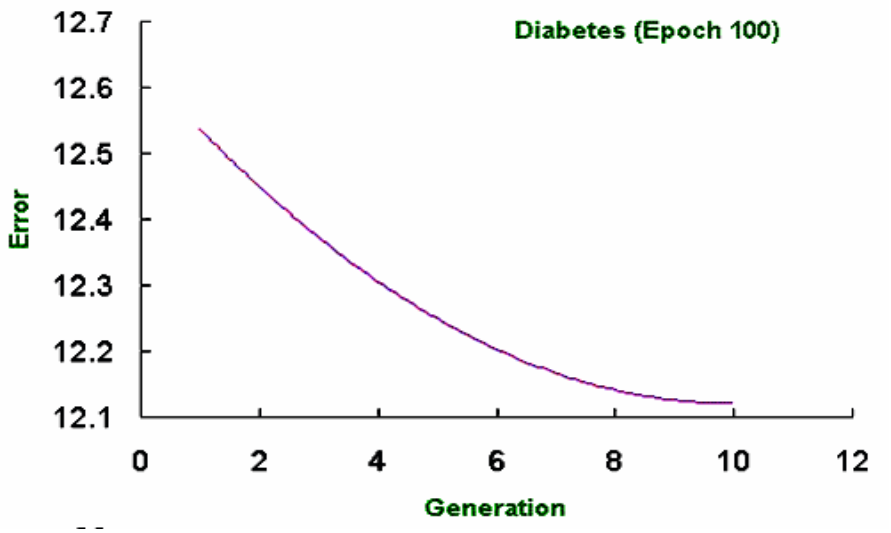

Figure 5.1: Generation Vs Error for diabetes.

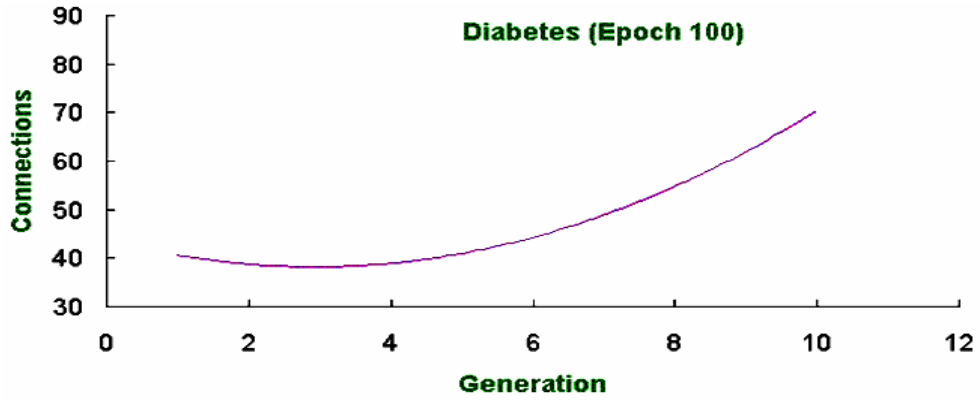

Figure 5.2: Generation Vs Connections for diabetes.

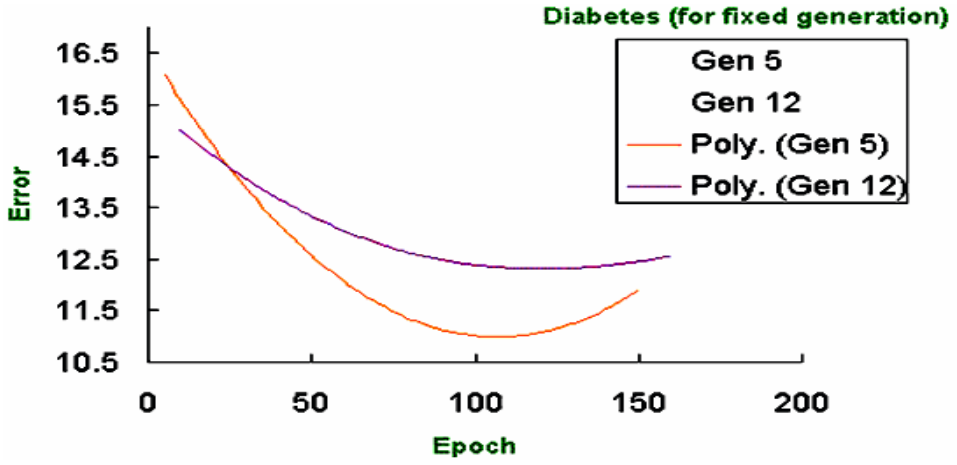

Figure 5.3: Epoch Vs Error for diabetes.

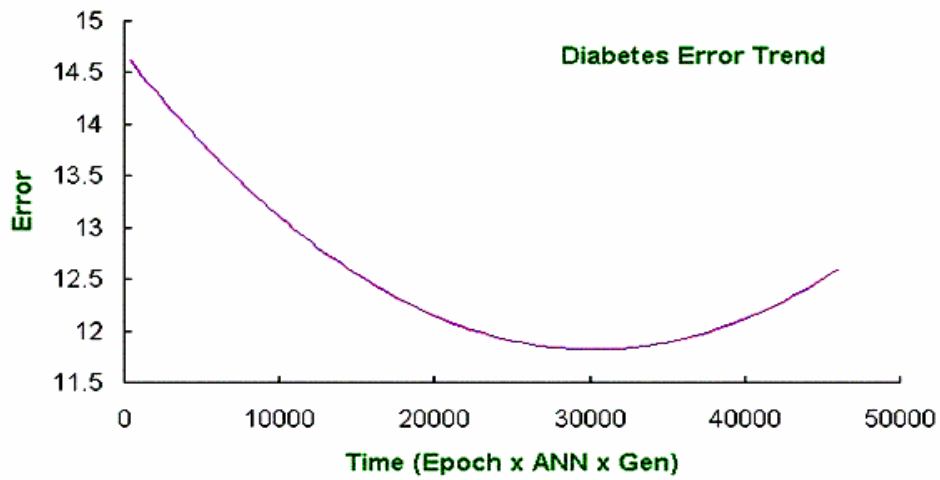

Figure 5.4: Time Vs Error for diabetes.

For the thyroid problem, as depicted in Figure 5.5, it is found the following trend of generation versus error, error is decreased as generation is increased up to certain level. Generation versus connections in Figure 5.6 shows that number of connection is minimum around generation 3. In Figure 5.7, epoch versus error for both generation 1 and 7 are shown. Figure 5.8 indicates, error is minimized when time is near 27000.

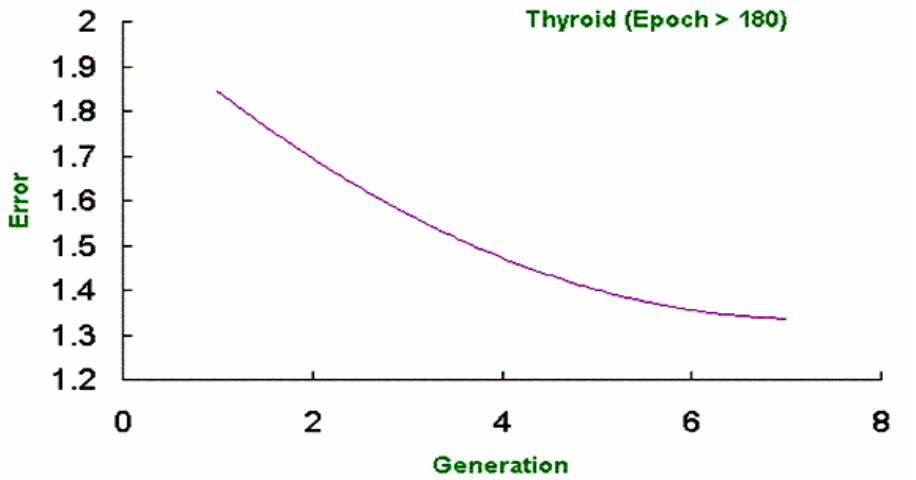

Figure 5.5: Generation Vs Error for Thyroid.

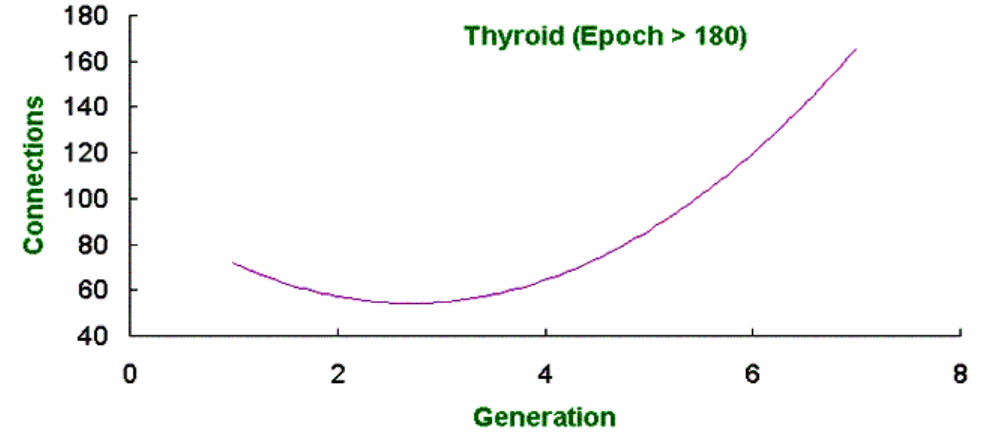

Figure 5.6: Generation Vs Connections for Thyroid.

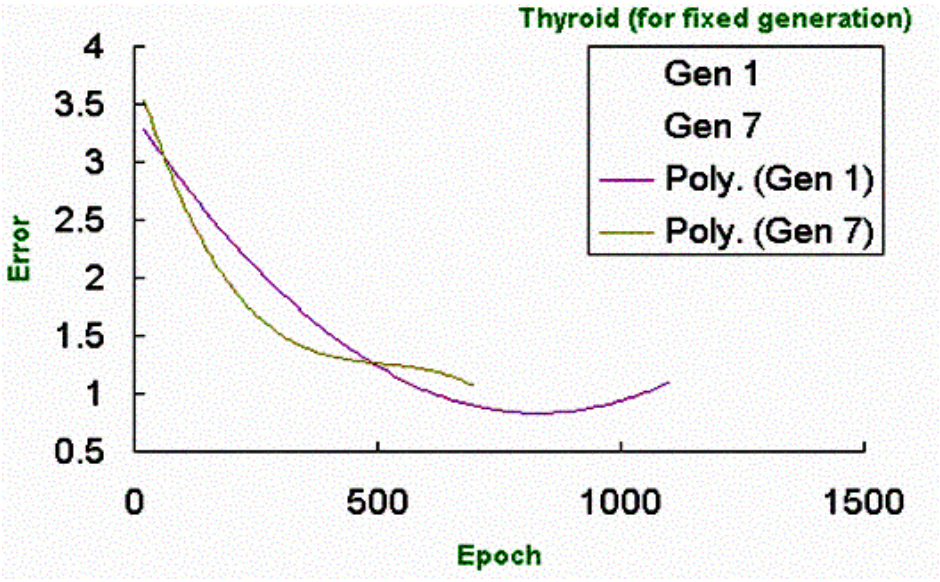

Figure 5.7: Epoch Vs Error for Thyroid.

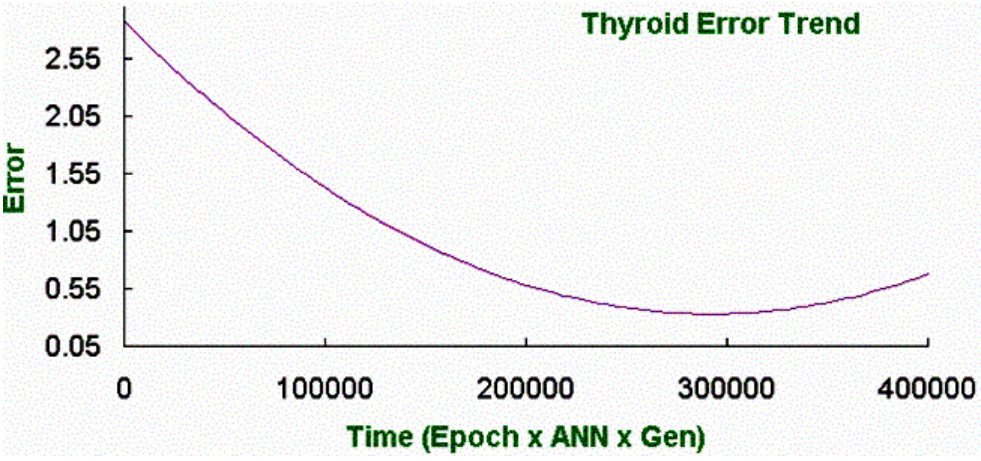

Figure 5.8: Time Vs Error for Thyroid.

For the heart disease problem the following trend of generation versus error graph in Figure 5.9 is found, error is decreased as generations is increased up to certain level. Generation versus connections in Figure 5.10 shows that number of connection is maximizing around generation 45. In Figure 5.11, epoch versus error for both generation 20 and 60 are shown. Figure 5.12 indicates, error is minimized when time is near 26000.

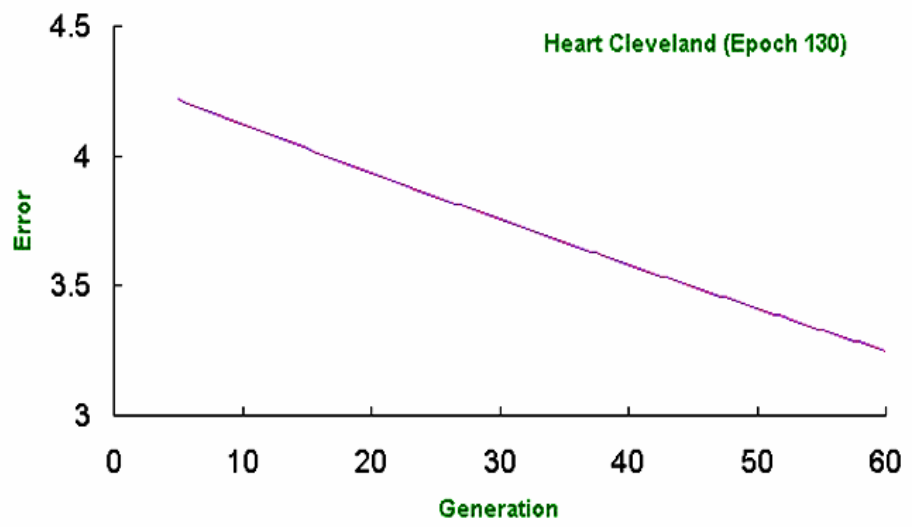

Figure 5.9: Generation Vs Error for heart disease.

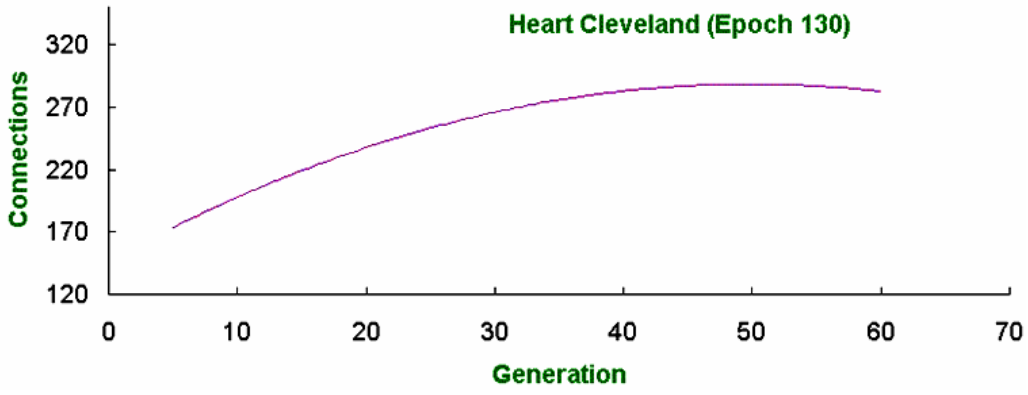

Figure 5.10: Generation Vs Connections for heart disease.

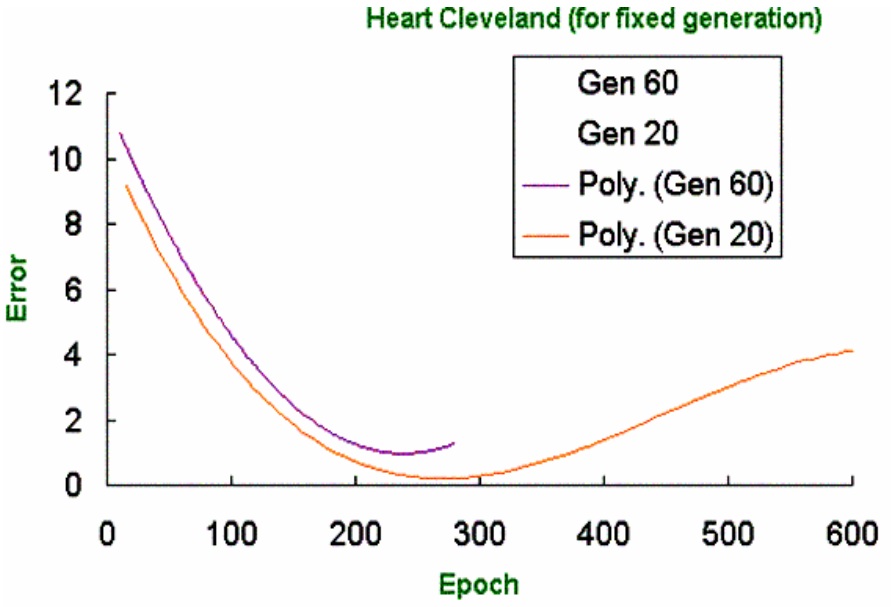

Figure 5.11: Epoch Vs Error for heart disease.

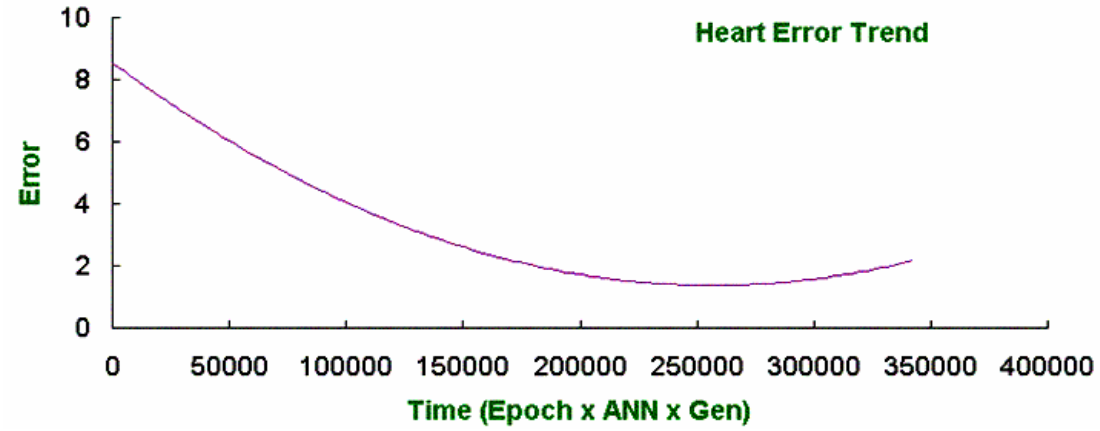

Figure 5.12: Time Vs Error for heart disease.

For the breast cancer problem the following trend of generation versus error in Figure 5.13 is found, error is decreased as generation is increased up to certain level. Generation versus connections in Figure 5.14 shows that number of connection is maximizing around generation 40. In Figure 5.15, epoch versus error for both generation 1 and 10 are shown. Figure 5.16 indicates error is minimized when time is near 22000.

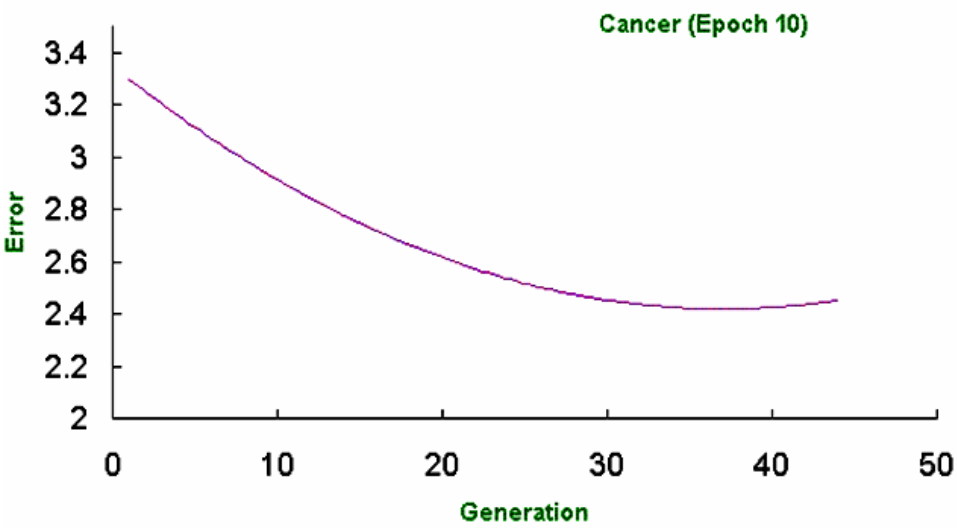

Figure 5.13: Generation Vs Error for breast cancer.

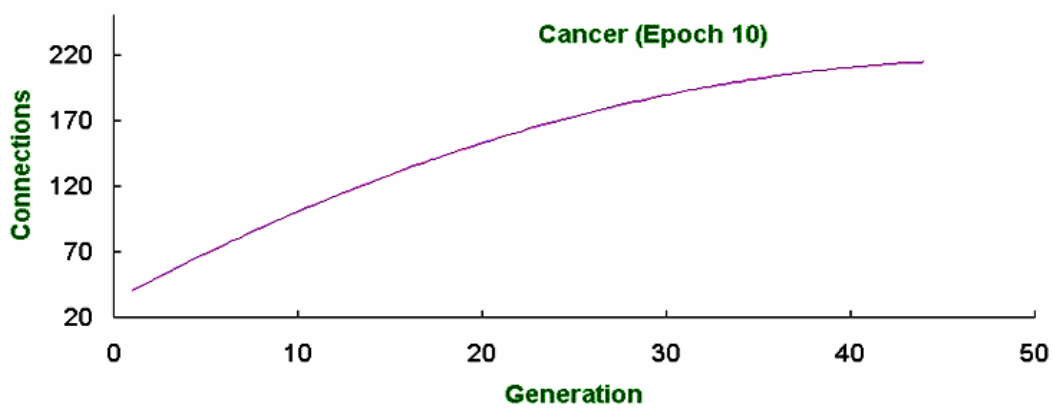

Figure 5.14: Generation Vs Connections for breast cancer.

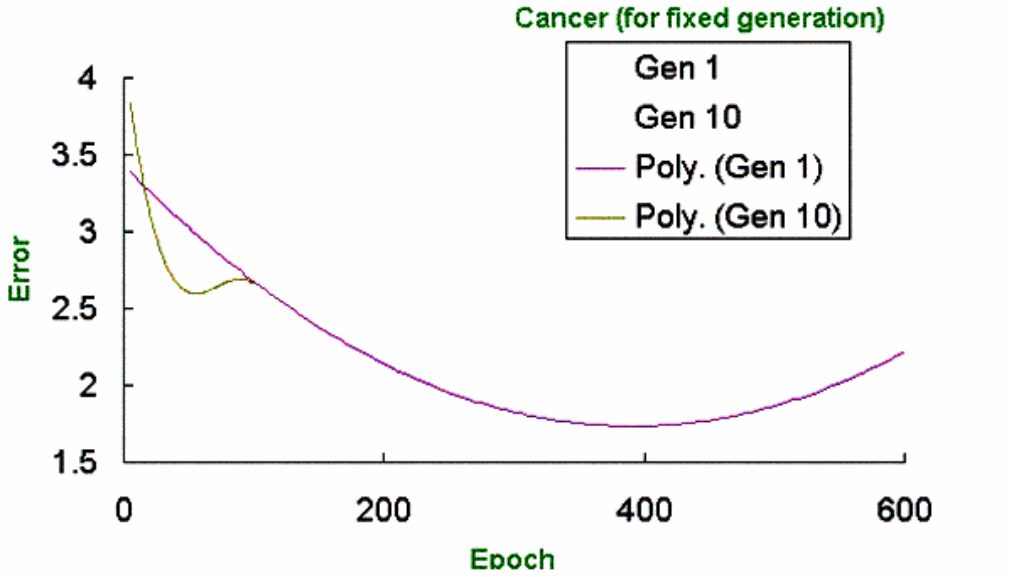

Figure 5.15: Epoch Vs Error for breast cancer.

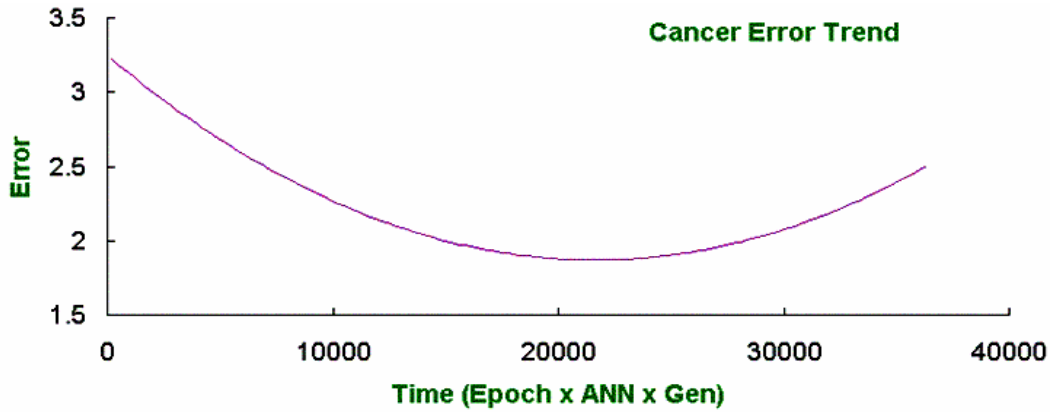

Figure 5.16: Time Vs Error for breast cancer.

Here is now given an example of ANN, both cellular encoding and direct encoding. It is taken the best (in term of least error rate) ANN found in breast cancer data set. The CE of this ANN is: PAR, PAR, SEQ, CUT 30, CUT 21, CUT 22, MRG 20, END, PAR, END, END, END, END. The program symbol tree is shown in Figure 5.17.

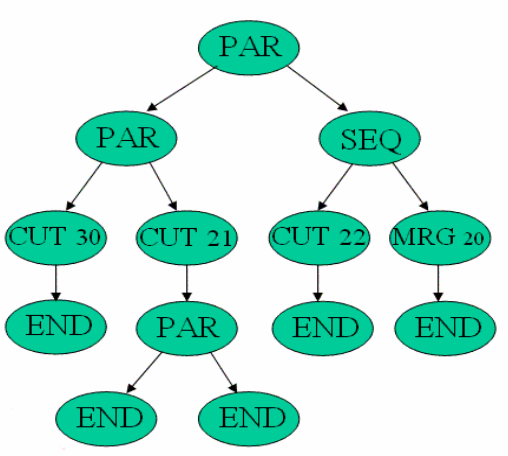

Figure 5.17: PST of the best ANN found in breast cancer problem.

The corresponding direct encoding is shown in Figure 5.18 too.

	0	1	2	3	4	5	6	7	8	9	0	1	2	0	1
0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
2	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0
5	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
6	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
7	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
8	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
9	2	1	0	0	0	0	0	0	0	0	0	0	0	1	1
10	0	2	1	0	0	0	0	0	0	0	0	0	0	1	1
11	2	1	0	0	0	0	0	0	0	0	0	0	0	1	1
12	2	1	0	0	0	0	0	0	0	0	0	0	0	1	1

Figure 5.18: DE of the best ANN found in breast cancer problem.

5.4 Comparison with other works

Direct comparison with other evolutionary approaches to designing ANN is very difficult due to the lack of such results. Instead, the best and latest results available in the literature, regardless of whether the algorithm used was an evolutionary, a BP or a statistical one, are used in the comparison. It is possible that some papers which should have been compared with were overlooked. However, the aim of this thesis is not to compare this algorithm exhaustively with all other algorithms.

First, the results are compared with EPNet [62] although the data sets used for some problems may differ. For example, for the heart disease problem EPNet does not used 27 disputable patterns, but here is used. For heart disease, thyroid and breast cancer problems, the partitions into training and testing data sets are not matched too. Yet, one can have a rough comparison among the experimental outcomes as shown in Table 5.2 through Table 5.5. Here the best result (* marked) found in this approach is for the ANN with least testing error rate. Whereas the best counted in EPNet was based on compactness of the evolved ANN.

Although EPNet can evolve very compact ANN which generalizes well, they come with the cost of additional computation time in order to perform search. The total time needed in the breast cancer problem, for example, by EPNet was very high. It could require roughly 109,000 epochs for a single run [62] whereas it is on average around 14,000 in this approach. Although, the actual time was less since few runs reached the maximal number of generations. Similar estimations can be applied to other problems.

Table 5.2: Comparison with EPNet for heart disease problem.

Heart Data Set		No. of Connections	No. of Hidden nodes	No. of Generations	No. of Epochs	Training Error	Testing Error rate
EPNet	Mean	92.6	4.1	193.3	500×2	10.708	0.16765
	SD	40.8	2.1	60.3	_	0.748	0.02029
	Best	**34**	**1**	**120**	_	**8.848**	**0.13235**
Our approach	Mean	204.033058	5.73553719	44.702479	117.809	6.2917089	0.13652889
	SD	100.599696	2.90335609	57.967756	169.049	3.3793083	0.05559175
	Best*	**145**	**4**	**5**	**130**	**4.54072**	**0.053333**

Table 5.3: Comparison with EPNet for diabetes problem.

Diabetes Data Set		No. of Connections	No. of Hidden nodes	No. of Generations	No. of Epochs	Training Error	Testing Error rate
EPNet	Mean	52.3	3.4	132.2	400×2	16.674	0.22379
	SD	16.1	1.3	48.7	_	0.294	0.00014
	Best	27	**1**	**100**	_	**16.092**	**0.19271**
Our approach	Mean	56.75555556	6.059259259	9.99259259	127.962	13.001760	0.2510803
	SD	23.90824084	2.278265126	12.30368	141.479	1.7322284	0.0390747
	Best*	**51**	**6**	**1**	**200**	**11.989235**	**0.208333**

Table 5.4: Comparison with EPNet for Thyroid problem.

Thyroid Data Set		No. of Connections	No. of Hidden nodes	No. of Generations	No. of Epochs	Training Error	Testing Error rate
EPNet	Mean	219.6	5.9	45.0	350×3	0.47	0.02115
	SD	74.36	2.4	12.5	_	0.091	0.00220
	Best	**128**	**3**	**10**	_	**0.336**	**0.01634**
Our approach	Mean	80.0882353	4.441176471	11.0882353	450.588	2.2820206	0.057467265
	SD	40.8429124	2.258870413	15.8733007	883.108	0.9273925	0.031933939
	Best*	**114**	**5**	**1**	**3500**	**0.672626**	**0.027222**

Table 5.5: Comparison with EPNet for breast cancer problem.

Cancer Data Set		No. of Connections	No. of Hidden nodes	No. of Generations	No. of Epochs	Training Error	Testing Error rate
EPNet	Mean	41.0	2.0	137.3	400×2	3.246	0.01376
	SD	14.7	1.1	37.7	_	0.589	0.00938
	Best	**15**	**0**	**100**	_	**1.544**	**0.00**
Our approach	Mean	80.3763441	7.77419355	10.2473118	70.5376	2.6638425	0.025806527
	SD	58.6144206	4.72304487	12.7156664	124.189	0.7204406	0.016328747
	Best*	**40**	**4**	**1**	**10**	**3.658594**	**0.005714**

It is also compared with the results of different other works (L Prechelt, W. Schiffman, R. Miranda, K. Bennet, O. L. Mangasarian, R. Werner , R. Reed, A. Roy, S. Govil etc). The following tables (Table 5.6 through Table 5.9) show them concisely. Here the evolutionary system proposed beats all other systems except for the thyroid data set as shown in Table 5.9.

For the heart disease problem, Table 5.6 shows results from this algorithm and other neural and non-neural algorithms. The GM algorithm [3] is used to construct RBF networks. It produced a RBF network of 24 Gaussians with 18.18% testing error. Bennet and Mangasarian [30] reported a testing error rate of 16.53% with their MSM1 method, 25.92% with their MSM method, and about 25% with BP, which is much worse than the worst ANN evolved here. The best manually designed ANN achieved 14.78% testing

error [34], which is worse than the best result of this approach, 5.33%. Again, Panayiota et. al. [39] found 16.8% training error rate and 27.4% testing error rate by their enhanced guided annealing technique.

Table 5.6: Comparison with other works for heart disease problem.

Heart Data Set	*RBF of GM algorithm*	*Bennet – Mangasarian (MSM1)*	*Bennet – Mangasarian (MSM)*	*Bennet – Mangasarian (BP)*	*Manually Designed ANN*	*Our approach*
Error rate of the Best ANN	18.18%	16.53%	25.92%	25%	14.78%	5.33%

For the breast cancer problem, Prechelt [34] reported results on manually constructed ANN (denoted as HDANN's) after testing a number of different ANN architectures. Table 5.7 shows that this approach found better ANN with error rate 0.5714% whereas it is 1.149% with HDANN. Again, Panayiota et.al. [39] found 2.35% training error rate and 4.7% testing error rate by their enhanced guided annealing technique. Ravi et. al. got it upto 95.5% accuracy level by their ALAR method [45].

Table 5.7: Comparison with other works for breast cancer problem.

Cancer Data Set	*HDANNS by trial and error*	*Our approach*
Error rate of the Best ANN	1.149%	0.5714%

The diabetes problem is one of the most challenging problems in ANN and machine learning due to its relatively small data set and high noise level. In the medical domain, data are often very costly to obtain. It would be unreasonable if an algorithm relies on more training data to improve its generalization. Table 5.8 compares the result with that one produced by Prechelt [34]. He found an ANN with eight hidden node which achieved the testing error rate of 0.2135 (21.35%), while ANN with the testing error rate of 0.2083 (20.83%) is achieved here outperforming his results.

Table 5.8: Comparison with other works for diabetes problem.

Diabetes Data Set	*Prechelt*	*Our approach*
Error rate of the Best ANN	**21.35%**	**20.83%**

Schiffmann et al. [59] tried the thyroid problem using a 21-20-3 network. They found that several thousand learning passes were necessary to achieve a testing error rate of 2.6% for this network. They also used their genetic algorithm to train multilayer ANN on the reduced training data set containing 713 examples. They obtained a network with testing error rate 2.5%. These results are slightly better than those generated by the ANN evolved by this approach. Table 5.9 summarizes the above results.

Table 5.9: Comparison with other works for thyroid problem.

Thyroid Data Set	*Schiffmann et al. 21-20-3 net*	*Schiffmann et al. Multilayer ANN*	*Our approach*
Error rate of the Best ANN	**2.6%**	**2.5%**	**2.7%**

Chapter 6

Conclusion & Future Research

6.1 Concluding Remarks

This thesis presents a new indirect encoding scheme, known as MCE, based on CE for evolving feedforward ANNs. The salient feature MCE is that it does not permutation problem of conventional crossover operator of Gas. A close and complete set of program symbols is chosen to generate the PSTs, i.e. the genotypes of ANNs. Some symbols of CE are excluded and the functionalities of other symbols are changed. New restrictions are also imposed on their appearances in the PSTs. These upgradations of CE result a permutation problem free encoding.

Consequently, one can employ the genetic operator crossover on the genotypes of ANNs in the evolutionary system, that is, the difficulties in producing highly fit offspring would no longer exist with crossover operators. Here, crossover tries all kinds of evolutions, i.e. deletion or addition of nodes and connections. Close behavioural link between the parents and their offspring is maintained by adopting a number of techniques. For example, partial training is always employed after each architectural change in order to reduce the behavioural disruption to an individual. To reduce the drastic change of architecture (and behaviour) from parents to their children, crossover is allowed at the lower levels of PSTs with higher probability. A hidden node is not added to an existing ANN at random, but through splitting an existing node by means of an additional program symbol to its PST. The proposed genetic search algorithm in this paper implements these strategies which imply significant improvement is the reduction of the number of user specified parameters. Since this approach searches a much larger space than that searched by most

other constructive or pruning algorithms and thus seems to require longer computation time, but it outperforms the time needed in other contemporary works.

6.2 Future Directions

In this subsection, four directions are given to extend the research followed in this thesis.

a) One of the important goals of the contemporary research going on the evolutionary artificial neural network is to reduce **evolution time**. In the evolutionary search training algorithm is not applied directly on the genotypes of the population. Rather a conversion of the cellular encoding to direct encoding is performed to learn current population through backpropagation. If it can be saved this conversion time by incorporating directly the cellular encodings with the training phase, then significant improvement in the generation time will be gained. Researchers are still waiting for an efficient training algorithm directly applicable over cellular encoded neural networks.

b) Another future research direction can be reducing the **number** of user defined parameters. Not only that, evolutionary parameters can be made **adaptive** with the search performance. Eiben et.al. [1] describe in details why it is necessary. As mentioned earlier, parameter tuning by hand is a common practice in evolutionary computation. Typically, one parameter is tuned at a time, which may cause some sub-optimal choices, since parameters are not independent and they often interact in a complex way. Simultaneous tuning of more parameters, however, leads to an enormous amount of experiments. Also, it is intuitive that different values of parameters might be optimal at different stages the dynamic evolutionary process. Hence adaptive (with respect to search stages / performances) parameters may lead to superior search result.

c) In order to reduce the noise in fitness evolution, the evolutionary system can evolve ANN **architectures and weights** simultaneously. Learned architectures and weights in one generation are inherited by the next generation. This is closer to the Lamarckian evolution than to the Darwinian one. Also, this is quite different from most genetic approaches where only architectures not weights are passed to the next generation [62].

d) To improve the rate of convergence in the training process for ANN one can follow the parallel nonlinear optimizing techniques proposed recently by Paul et. al. [40], which

ultimately will speed up the evolutionary search. Also, ANN task decomposition method can be adopt based on output parallelism [51] to increase learning speed. Divide and conquer (DCL) scheme by Hsin et. al. [26] and the suggestions for associative memories proposed by Yingquan et. al. [63] can also be considered.

e) One of the future improvements would be giving more attention to the **compactness** of the evolved ANN. For example, the EPNet algorithm of Xin Yao et. al. [62] produces very compact ANN which is an attractive property of their evolutionary approach. But this is achieved at the cost of longer computation time. Thus, if a new evolutionary system can be proposed that encourages the parsimony of the evolved ANN without compromising the evolution time, it will be a very appreciable.

Bibliography

[1] Ágoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz, “Parameter control in evolutionary algorithms”, *IEEE Trans. on Evolutionary Computation,* vol. 3, pp: 124-141, 1999.

[2] Aristid Lindenmayer, “Mathematical Models for Cellular Interactions in Development”, in *Journal of Theoretical Biology*, vol. 18, pp. 280-299, 1968.

[3] A. Roy, S. Govil, and R. Miranda, “An algorithm to generate radial basis function (RBF)-like nets for classification problems,” *Neural Networks*, vol. 8, pp. 179–201, 1995.

[4] Avelino J. Gonzalez, and Douglas D. Dankel, "The Engineering of Knowledge-based Systems", *Prentice-Hall Inc.* ISBN 0-13-334293-X, 1993.

[5] Bart L. M. Happel, and Jacob M. J. Murre, “Design and Evolution of Modular Neural Network Architectures”, *in Neural Networks*, vol. 7, no. 6/7, pp. 985-1004, 1994.

[6] C. M. Friedrich and C. Moraga, “An evolutionary method to find good building blocks for architectures of artificial neural networks”, *Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems*, pp. 951-956, 1996.

[7] Christian Jacob, and Jan Rehder, “Evolution of Neural Net Architectures by a Hierarchical Grammar-based Genetic System”, in *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, Innsbruck, pp. 72-79, 1993.

[8] Data and Analysis Center for Software, "Artificial Neural Networks Technology", *http://www.dacs.dtic.mil/techs/neural/neural.title.html*, Rome. NY, August, 1992.

[9] David W. White, “GANNet: A genetic Algorithm for Searching Topology and Weight Spaces in Neural Network Design”, *Dissertation at the University of Maryland*, 1993.

[10] David E. Goldberg, “Genetic algorithms in search, optimization, and machine learning”, published by *Pearson Education (Singapore) Pte Ltd*, ISBN 81-7808-130-X, Fifth Indian Reprint, pp. 10-14, pp. 236-238, 2002.

[11] David J. Montana, “Automated Parameter Tuning for Interpretation of Synthetic Images”, in the *Handbook for Genetic Algorithms*, pp. 282-311, 1991.

[12] D. B. Fogel, “Evolutionary computation: towards a new philosophy of machine intelligence”, *IEEE Press*, NY 10017-2394, 1995.

[13] D. Montana, and L. Davis, “Training Feedforward Neural Networks using Genetic Algorithms”, in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 762-767, 1989.

[14] Darell Whitley, J. David Schaffer, and Larry J. Eshelman, “Combinations of genetic algorithms and neural networks: A survey of the State of the Art”, in *Proceedings of the International Workshop on Combinations of genetic algorithms and neural networks*, Baltimore, IEEE, pp. 1-37, 1992.

[15] D. Whitley, T. Starkweather, and C. Bogart, “Genetic algorithms and neural networks: optimizing connections and connectivity”, in *Parallel Computing* 14, North-Holland, pp. 347-361, 1990.

[16] Egber Boers, and Herman Kuiper, “Biological Metaphors and the Design of Modular Artificial Neural Networks”, *Master thesis at Leiden University* , the Netherlands, 1992.

[17] Fatemeh Zahedi, "Intelligent Systems for Business: Expert Systems with Neural networks”, *Wadsworth Inc*. ISBN 0-534-18888-5, 1993.

[18] Frédéric Gruau, “Neural network synthesis using cellular encoding and the genetic algorithm”, *Ph.D. Thesis*, Ecole Normale Supérieure de Lyon, 1994.

[19] Frédéric Gruau and Darrell Whitley, “Adding learning to the cellular development of neural networks: evolution and the baldwin effect”, *Evolutionary Computation*, vol. 1, pp. 213-233, 1993.

[20] Frédéric Gruau, Darrell Whitley and Larry Pyeatt, “A comparison between cellular encoding and direct encoding for genetic neural networks”, *Proceedings of the First Genetic Programming Conference*, pp. 81-89, 1996.

[21] F. Kursawe, "Evolution Strategies: Simple Models of Natural Processes", *Revue Internationale De Systemique*, France, 1994.

[22] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hedge, “Designing Neural Networks using Genetic Algorithms”, in *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 379-384, 1989.

[23] Hiroaki Kitano, “Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems”, in *Complex Systems*, no. 4, pp. 461-476, 1990.

[24] Hiroaki Kitano, “Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms”, in *Eighth National Conference on Artificial Intelligence*, AAAI, MIT Press, vol. II, pp 789-795, 1990.

[25] H. Schwefel, "Collective Phaenomena in Evolutionary Systems", *31st Annual Meet. Inter'l Soc. for general system research*, Budapest, 1025-1033, 1987.

[26] Hsin-Chia Fu, Yen-Po Lee, Cheng-Chin Chiang and Hsiao-Tien Pao, "Divide and Conquer Learning and Modular Perceptron Networks", *IEEE transactions on neural networks*, vol 12, no 2, pp 250, March 2001.

[27] John R. Koza, and James P. Rice, "Genetic Generation of Both the Weight and Architecture for a Neural Network", in *Proceedings of the International Joint Conference on Neural Networks*, IEEE, vol. II, pp. 397-404, 1991.

[28] J. M. Bishop, and M. J. Bushnell, "Genetic Optimization of Neural Network Architectures for Colour Recipe Prediction", in *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, Innsbruck, pp. 719-725, 1993.

[29] J. P. Nadal, "Study of growth algorithm for a feedforward network", *International Journal of Neural Systems*, vol. 1, pp. 55-59, 1989.

[30] K. P. Bennet, and O. L. Mangasarian, "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets," *Optimization Methods Software*, vol. 1, pp. 23–34, 1992.

[31] Leonardo Marti, "Genetically Generated Neural Networks II: Searching for an Optimal Representation", in *IEEE International Joint Conference on Neural Networks*, vol. II, p. 221-226, 1992.

[32] L. Fausett, "Fundamentals of Neural Networks", *Englewood Cliffs*, Prentice-Hall Inc., pp. 461, 1994.

[33] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial intelligence through simulated evolutionary", New York, *NY: John Wiley & Sons*, 1996.

[34] L. Prechelt, “Proben1—A set of neural network benchmark problems and benchmarking rules,” *Fakultat fur Informatik*, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, Sept. 1994.

[35] Marko Grönroos, “A comparison of some methods for evolving neural networks”, *Proceedings of GECCO'99*, Morgan Kaufmann Publishers, San Francisco, California, vol. 2, 1999.

[36] Martin Mandischer, “Representation and Evolution of Neural Networks”, in *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, Innsbruck, pp. 643-649, 1993.

[37] Md. Monirul Islam, Xin Yao, Kazayuki Murase, “A Constructive Algorithm for Training Cooperative Neural Network Ensembles”, *IEEE transactions on neural networks*, vol 14, no 4, pp 820, July 2003.

[38] N. Burgess, “A constructive algorithm that converges for real-valued input patterns”, *International Journal of Neural Systems*, vol. 5, no. 1, pp. 59-66, 1994.

[39] Panayiota Poirazi, Costas Neocleous, Costantinos S. Pattichis and Cristos N. Schizas, “Classification Capacity of a Modular Neural Network Architecture and Implementing Neurally Inspired Architecture and Training Rules”, *IEEE transactions on neural networks*, vol 15, no 3, pp 597, May 2004.

[40] Paul K. H. Phua, Daohua Ming, “Parallel Nonlinear Optimization Techniques for Training Neural Networks”, *IEEE transactions on neural networks*, vol 14, no 6, pp 1460, Nov 2003.

[41] Philipp Köhn, “Genetic encoding strategies for neural networks”, *Master’s thesis*, University of Tennessee, Knoxville, IPMU, 1996.

[42] Petri Hodju, and Jokko Halme, “Neural Networks Information Homepage”, *http://koti.mbnet.fi/~phodju/nenet/index.html*, Copyright (c) 1999.

[43] P. J. Angeline, G. M. Sauders and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks”, *IEEE Trans. on Neural Networks*, vol. 5, no. 1, pp. 54-65, 1994.

[44] P. J. B. Hancock, “Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification”, *Proc. of the Int’l Workshop on Combinations of Genetic Algorithms and Neural Networks* (COGANN-92) (D. Whitley and J. D. Schaffer, eds.), IEEE Computer Society Press, Los Alamitos, CA, pp. 108-122,1992.

[45] Ravi Kothari and Vivek Jain, “Learning from labeled and unlabeled data using a minimal number of queries”, *IEEE transactions on neural networks*, vol 14, no 6, pp 1496, Nov 2003.

[46] Resonance Publications, Inc, “Neural Networks”, *http://www.Resonancepub. com neuralnets.htm*, June, 1998.

[47] R. K. Belew, J. McInerney, and N. N. Schraudolph, “Evolving networks: Using genetic algorithm with connectionist learning,” *Computer Sci. Eng. Dept.*, Univ. California-San Diego, Tech. Rep. CS90-174 revised, Feb. 1991.

[48] R. Reed, “Pruning algorithm – a survey”, *IEEE Trans. on Neural Networks*, vol. 4, no. 5, pp. 740-747, 1993.

[49] R. Setiono and L. C. K. Hui, “Use of a quasi-newton method in a feedforward neural network construction algorithm”, *IEEE Trans. on Neural Networks*, vol. 6, no. 1, pp. 273-277, 1995.

[50] S. E. Fahlman and C. Lebiere, “The cascade-correlation learning architecture”, *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), Morgan Kaufmann, San Mateo, CA, pp. 524-532, 1990.

[51] Sheng Uei Guan and Shanchun Li, “Parallel growing and training of neural networks using output parallelism”, *IEEE transactions on neural networks*, vol 13, no 3, pp 542, May 2002.

[52] Simon Haykin, “Neural networks: a comprehensive foundation”, published by *Prentice Hall International, Inc.*, Upper Saddle River, New Jersey 07458, ISBN 0-13-908385-5, pp. 161-175, 1999.

[53] Steven Alex Harp, and Tariq Samad, “Genetic Synthesis of Neural Network Architecture”, in *Handbook of Genetic Algorithms*, pp. 202-221, 1991.

[54] Steven Alex Harp, Tariq Samad, and Aloke Guha, “Towards the Genetic Synthesis of Neural Networks”, in *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 360-369, 1989.

[55] Talib Hussain, “Cellular encoding: review and critique”, *Queen’s University*, July 19, 1997.

[56] Vittorio Maniezzo, “Searching among Search Spaces: hastening the genetic evolution of feed-forward neural networks”, in *International Joint Conference on Neural Networks and Genetic Algorithms*, Innsbruck, pp. 635-642, 1993.

[57] Vittorio Maniezzo, “Genetic Evolution of the Topology and Weight Distribution of Neural Networks”, *in IEEE Transactions of Neural Networks*, vol. 5, No. 1, pp 39-53, 1994.

[58] Wolfram Schiffmann, Merten Joost, and Randolf Werner, “Performance Evaluation of Evolutionary Created Neural Network Topologies”, in *Parallel Problem Solving from Nature 2*, H.P. Schwefel and R. Maenner, Springer Verlag, pp. 292-296, 1991.

[59] W. Schiffmann, M. Joost, and R. Werner, “Synthesis and Performance Analysis of Neural Network Architectures”, Technical Report 16, *University of Koblenz, Germany*, ftp://archive.cis.ohio-state.edu(128.146.8.52)/pub/neuroprose/schiff.nnga.ps.Z, 1992.

[60] W. Schiffmann, M. Joost, and R. Werner, “Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons”, in *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, Innsbruck, pp. 675-682, 1993.

[61] W. S. Sarle, “Introduction. Periodic posting to the Usenet newsgroup”, *URL: ftp://ftp.sas.com/pub/neural/FAQ.html*, Neural Network FAQ, part 1 of 7, 1999.

[62] Xin Yao and Yong Liu, “A new evolutionary system for evolving artificial neural networks”, *IEEE transactions on neural networks*, vol 8, no 3, pp 694-713, 1997.

[63] Yingquan Wu and Stella N. Batalama, “An efficient learning algorithm for associative memories”, *IEEE transactions on neural networks*, vol 11, no 5, pp 1058, Sept 2000.

Appendix A

Neural Network Glossary

In this section some of the common terms about neural networks and genetic evolution, which have not discussed in the previous chapters, are given for the interested reader.

Activation / Initialization function: The time-varying value that is the output of a neuron.

Artificial Intelligence: An interdisciplinary approach to understanding human intelligence that has its common thread the computer as an experimental vehicle.

Associative memory: Also called `content-addressable' memory. This type of memory is not stored on any individual neuron but is a property of the whole network. It is by inputting to the network part of the memory. This is very different from conventional computer memory where a given memory (or piece of data) is assigned a unique address which is needed to recall that memory.

Baldwin effect: In hybrid strategies, the effect of using the individual's fitness determined by the objective function value after application of a local search. The individual's genotype serves as initial condition of the local search. However, unlike Lamarckian evolution, the individual's genotype remains unchanged.

Bias: The net input (or bias) is proportional to the amount that incoming neural activations must exceed in order for a neuron to fire.

Connectivity: The amount of interaction in a system, the structure of the weights in a neural network, or the relative number of edges in a graph.

Elitism/ elitist selection: Property of selection methods, which guarantees the survival of the best individual(s).

Encode network: A perceptron network designed to illustrate that the hidden layer nodes play a crucial role in allowing the network to learn about special features in the input patterns. Once it has learnt about the `generalized' features of the training pattern sit it can respond usefully in new situations.

Epoch: One complete presentation of the training set to the network during training.

Fitness: Evaluation of an individual with respect to its reproduction capability. Selection in EA is based on the fitness. Generally, it is determined on the basis of the objective value(s) of the individual in comparison with all other individuals in the selection pool. The fitness function may additionally depend on different side conditions/constraints and stochastic influences (fitness noise/noisy fitness). The term ``fitness function'' is often used as a synonym for objective function. It varies greatly from one type of program to the next. For example, if one were to create a genetic program to set the time of a clock, the fitness function would simply be the amount of time that the clock is wrong. Unfortunately, few problems have such an easy fitness function; most cases require a slight modification of the problem in order to find the fitness.

Generalization: A measure of how well a network can respond to new images on which it has not been trained but which are related in some way to the training patterns. An ability to generalize is crucial to the decision making ability of the network.

Genotype: In EA with genotype-phenotype mapping, the genotype is the representation on which the crossover and mutation operators are applied to (see also phenotype).

Hopfield network: A particular example of an artificial neural network capable of storing and recalling memories or patterns. All nodes in the network feed signals to all others.

Input layer: Neurons whose inputs are fed from the outside world.

Lamarckian evolution: Adjustment of the genotype to the locally optimized offspring (local search) in hybrid strategies.

Layer: A group of neurons that have a specific function and are processed as a whole. The most common example is in a feedforward network that has an input layer, an output layer and one or more hidden layers.

Learning parameter: Also learning rate, in self-adaptive ES/EP, an exogenous strategy parameter which influences the speed of self-adaptation of the mutation strength

Linear Networks: A general scientific principal is that a simple model should always be chosen in preference to a complex model if the latter does not fit the data better. In terms of function approximation, the simplest model is the linear model, where the fitted function is a hyperplane. In classification, the hyperplane is positioned to divide the two classes (a linear discriminant function); in regression, it is positioned to pass through the data. A linear model is typically represented using an $N \times N$ matrix and an $N \times 1$ bias vector.

A neural network with no hidden layers, and an output with dot product synaptic function and identity activation function, actually implements a linear model. The weights correspond to the matrix, and the thresholds to the bias vector. When the network is executed, it effectively multiplies the input by the weights matrix then adds the bias vector.

The linear network provides a good benchmark against which to compare the performance of your neural networks. It is quite possible that a problem that is thought to be highly complex can actually be solved as well by linear techniques as by neural

networks. If you have only a small number of training cases, you are probably anyway not justified in using a more complex model.

Multilayer-perceptron (MLP): This is perhaps the most popular network architecture in use today, due originally to Rumelhart and McClelland (1986) and discussed at length in most neural network textbooks (e.g., Bishop, 1995). MLP is a type of feedforward neural network that is an extension of the perceptron in that it has at least one hidden layer of neurons. Layers are updated by starting at the inputs and ending with the outputs. Each neuron computes a weighted sum of the incoming signals, to yield a net input, and passes this value through its sigmoidal activation function to yield the neuron's activation value. Unlike the perceptron, an MLP can solve linearly inseparable problems. A graphical representation of an MLP is shown below.

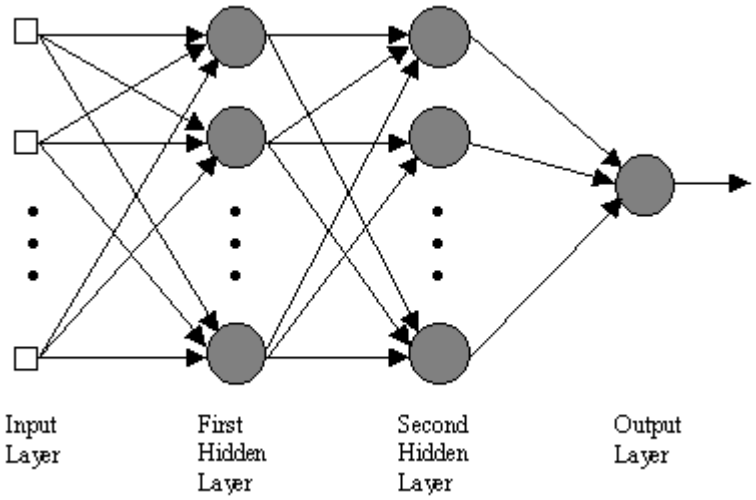

Figure A.1: A multilayer perceptron.

Neuron: A simple computational unit that performs a weighted sum on incoming signals, adds a threshold or bias term to this value to yield a net input, and maps this last value through an activation function to compute its own activation. Some neurons, such as those found in feedback or Hopfield networks, will retain a portion of their previous activation.

Output neuron: A neuron within a neural network whose outputs are the result of the network.

Over-learning and Generalization: One major problem with the approach outlined above is that it doesn't actually minimize the error that one is really interested in - which is the expected error the network will make when new cases are submitted to it. In other words, the most desirable property of a network is its ability to generalize to new cases. In reality, the network is trained to minimize the error on the training set, and short of having a perfect and infinitely large training set, this is not the same thing as minimizing the error on the real error surface - the error surface of the underlying and unknown model. The most important manifestation of this distinction is the problem of over-learning, or over-fitting.

How can one select the right complexity of network? A larger network will almost invariably achieve a lower error eventually, but this may indicate over-fitting rather than good modeling. The answer is to check progress against an independent data set, the selection set. Some of the cases are reserved, and not actually used for training in the back propagation algorithm. Instead, they are used to keep an independent check on the progress of the algorithm. It is invariably the case that the initial performance of the network on training and selection sets is the same (if it is not at least approximately the same, the division of cases between the two sets is probably biased). As training progresses, the training error naturally drops, and providing training is minimizing the true error function, the selection error drops too. However, if the selection error stops dropping, or indeed starts to rise, this indicates that the network is starting to overfit the data, and training should cease. When over-fitting occurs during the training process like this, it is called over-learning. In this case, it is usually advisable to decrease the number of hidden units and/or hidden layers, as the network is over-powerful for the problem at hand. In contrast, if the network is not sufficiently powerful to model the underlying function, over-learning is not likely to occur, and neither training nor selection errors will drop to a satisfactory level.

Perceptron: An artificial neural network capable of simple pattern recognition and classification tasks. It is composed of three layers where signals only pass forward from nodes in the input layer to nodes in the hidden layer and finally out to the output layer. There are no connections within a layer.

Phenotype: Expression of the properties coded by the individual's genotype. The expression/development of the phenotype can additionally be influenced by (stochastic) constraints. The precise definition is mostly problem-dependent. For parameter optimization the phenotype is usually identical with the object parameters, whereas for structure optimization (e.g. of neural networks) the phenotype represents a specific structure.

Population: Pool of individuals exhibiting equal or similar genome structures, which allows the application of genetic operators

Probabilistic Neural Networks: A useful interpretation of network outputs was as estimates of probability of class membership, in which case the network was actually learning to estimate a probability density function (p.d.f.). A similar useful interpretation can be made in regression problems if the output of the network is regarded as the expected value of the model at a given point in input-space. This expected value is related to the joint probability density function of the output and inputs.

Estimating probability density functions from data has a long statistical history (Parzen, 1962), and in this context fits into the area of Bayesian statistics. Conventional statistics can, given a known model, inform us what the chances of certain outcomes are (e.g., it is known that an unbiased die has a 1/6th chance of coming up with a six). Bayesian statistics turns this situation on its head, by estimating the validity of a model given certain data. More generally, Bayesian statistics can estimate the probability density of model parameters given the available data. To minimize error, the model is then selected whose parameters maximize this p.d.f.

In the context of a classification problem, if one can construct estimates of the p.d.f.s of the possible classes, one can compare the probabilities of the various classes, and select

the most-probable. This is effectively what one ask a neural network to do when it learns a classification problem - the network attempts to learn (an approximation to) the p.d.f.

A more traditional approach is to construct an estimate of the p.d.f. from the data. The most traditional technique is to assume a certain form for the p.d.f. (typically, that it is a normal distribution), and then to estimate the model parameters. The normal distribution is commonly used as the model parameters (mean and standard deviation) can be estimated using analytical techniques. The problem is that the assumption of normality is often not justified.

An alternative approach to p.d.f. estimation is kernel-based approximation (Parzen, 1962; Speckt, 1990; Speckt, 1991; Bishop, 1995; Patterson, 1996). One can reason loosely that the presence of particular case indicates some probability density at that point: a cluster of cases close together indicate an area of high probability density. Close to a case, one can have high confidence in some probability density, with a lesser and diminishing level as one moves away. In kernel-based estimation, simple functions are located at each available case, and added together to estimate the overall p.d.f. Typically, the kernel functions are each Gaussians (bell-shapes). If sufficient training points are available, this will indeed yield an arbitrarily good approximation to the true p.d.f.

This kernel-based approach to p.d.f. approximation is very similar to radial basis function networks, and motivates the probabilistic neural network (PNN) and generalized regression neural network (GRNN), both devised by Speckt (1990 and 1991). PNNs are designed for classification tasks and GRNNs for regression. These two types of network are really kernel-based approximation methods cast in the form of neural networks.

In the PNN, there are at least three layers: input, radial, and output layers. The radial units are copied directly from the training data, one per case. Each models a Gaussian function centered at the training case. There is one output unit per class. Each is connected to all the radial units belonging to its class, with zero connections from all other radial units. Hence, the output units simply add up the responses of the units belonging to their own class. The outputs are each proportional to the kernel-based estimates of the p.d.f.s of the various classes, and by normalizing these to sum to 1.0 estimates of class probability are produced.

The greatest advantages of PNNs are the fact that the output is probabilistic (which makes interpretation of output easy), and the training speed. Training a PNN actually consists mostly of copying training cases into the network, and so is as close to instantaneous as can be expected.
The greatest disadvantage is network size: a PNN network actually contains the entire set of training cases, and is therefore space-consuming and slow to execute.
PNNs are particularly useful for prototyping experiments (for example, when deciding which input parameters to use), as the short training time allows a great number of tests to be conducted in a short period of time.

Radial Basis Function Networks: MLP models response functions using the composition of sigmoid-cliff functions - for a classification problem, this corresponds to dividing the pattern space up using hyperplanes. The use of hyperplanes to divide up space is a natural approach - intuitively appealing, and based on the fundamental simplicity of lines. An equally appealing and intuitive approach is to divide up space using circles or (more generally) hyperspheres. A hypersphere is characterized by its center and radius. More generally, just as an MLP unit responds (non-linearly) to the distance of points from the line of the sigmoid-cliff, in a radial basis function network (Broomhead and Lowe, 1988; Moody and Darkin, 1989; Haykin, 1994) units respond (non-linearly) to the distance of points from the center represented by the radial unit. The response surface of a single radial unit is therefore a Gaussian (bell-shaped) function, peaked at the center, and descending outwards. Just as the steepness of the MLP's sigmoid curves can be altered, so can the slope of the radial unit's Gaussian. MLP units are defined by their weights and threshold, which together give the equation of the defining line, and the rate of fall-off of the function from that line. Before application of the sigmoid activation function, the activation level of the unit is determined using a weighted sum, which mathematically is the dot product of the input vector and the weight vector of the unit; these units are therefore referred to as dot product units. In contrast, a radial unit is defined by its center point and a radius. A point in N dimensional space is defined using N numbers, which exactly corresponds to the number of weights in a dot product unit, so the center of a radial unit is stored as weights. The radius (or deviation)

value is stored as the threshold. It is worth emphasizing that the weights and thresholds in a radial unit are actually entirely different to those in a dot product unit, and the terminology is dangerous if you don't remember this: Radial weights really form a point, and a radial threshold is really a deviation.

A radial basis function network (RBF), therefore, has a hidden layer of radial units, each actually modeling a Gaussian response surface. Since these functions are nonlinear, it is not actually necessary to have more than one hidden layer to model any shape of function: sufficient radial units will always be enough to model any function. The remaining question is how to combine the hidden radial unit outputs into the network outputs? It turns out to be quite sufficient to use a linear combination of these outputs (i.e., a weighted sum of the Gaussians) to model any nonlinear function. The standard RBF has an output layer containing dot product units with identity activation function.

RBF networks have a number of advantages over MLPs. First, as previously stated, they can model any nonlinear function using a single hidden layer, which removes some design-decisions about numbers of layers. Second, the simple linear transformation in the output layer can be optimized fully using traditional linear modeling techniques, which are fast and do not suffer from problems such as local minima which plague MLP training techniques. RBF networks can therefore be trained extremely quickly (i.e., orders of magnitude faster than MLPs).

Experience indicates that the RBF's more eccentric response surface requires a lot more units to adequately model most functions. Of course, it is always possible to draw shapes that are most easily represented one way or the other, but the balance does not favor RBFs. Consequently, an RBF solution will tend to be slower to execute and more space consuming than the corresponding MLP (but it was much faster to train, which is sometimes more of a constraint). RBFs are also more sensitive to the curse of dimensionality, and have greater difficulties if the number of input units is large.

Self-organizing: A network is called self-organizing if it is capable of changing its connections so as to produce useful responses for input patterns without the instruction of a smart teacher.

Sigmoid function: An S-shaped function that is often used as an activation function in a neural network.

SOFM Networks: Self Organizing Feature Map (SOFM, or Kohonen) networks are used quite differently to the other networks. Whereas all the other networks are designed for supervised learning tasks, SOFM networks are designed primarily for unsupervised learning (see Kohonen, 1982; Haykin, 1994; Patterson, 1996; Fausett, 1994). A SOFM network has only two layers: the input layer, and an output layer of radial units (also known as the topological map layer). The units in the topological map layer are laid out in space - typically in two dimensions. SOFM networks are trained using an iterative algorithm. Once the network has been trained to recognize structure in the data, it can be used as a visualization tool to examine the data.

Threshold: A quantity added to (or subtracted from) the weighted sum of inputs into a neuron, which forms the neuron's net input. Intuitively, the net input (or bias) is proportional to the amount that the incoming neural activations must exceed in order for a neuron to fire.

Weight: In a neural network, the strength of a synapse (or connection) between two neurons. Weights may be positive (excitatory) or negative (inhibitory). The thresholds of a neuron are also considered weights, since they undergo adaptation by a learning algorithm.