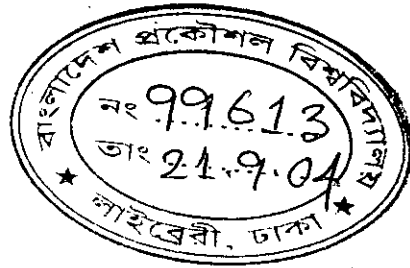


Evolving Artificial Neural Networks Using Permutation Problem Free Modified Cellular Encoding

by

Mohammad Masud Hasan



Submitted for the partial fulfillment of the requirements for the degree of
M.Sc. Engineering in Computer Science and Engineering

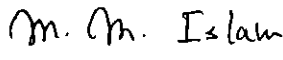
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh

August 16, 2004

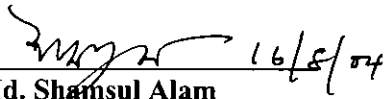


The thesis, EVOLVING ARTIFICIAL NEURAL NETWORKS USING PERMUTATION PROBLEM FREE MODIFIED CELLULAR ENCODING, submitted by MOHAMMAD MASUD HASAN, ROLL No. 040205035P, Session: April, 2002, Registration No. 95394 to the Department of Computer Science and Engineering of Bangladesh University of Engineering and Technology has been accepted as satisfactory for partial fulfillment of the requirements for the degree of M.Sc. Engg. in Computer Science and Engineering and approved as to its style and contents. Examination held on August 16, 2004.

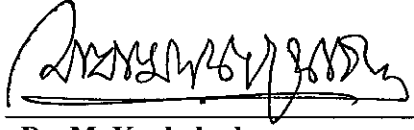
BOARD OF EXAMINERS

- 1 

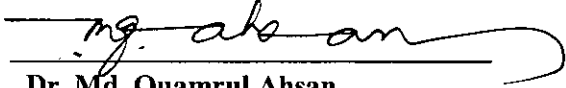
Dr. Md. Monirul Islam
Assistant Professor
Department of CSE
BUET, Dhaka-1000 **Chairman
(Supervisor)**

- 2 

Dr. Md. Shamsul Alam
Professor and Head
Department of CSE
BUET, Dhaka-1000 **Member
(Ex-officio)**

- 3 

Dr. M. Kaykobad
Professor
Department of CSE
BUET, Dhaka-1000 **Member**

- 4 

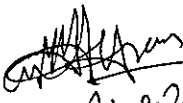
Dr. Md. Quamrul Ahsan
Professor
Department of EEE
BUET, Dhaka-1000 **Member
(External)**

Declaration

I, hereby, declare that the work presented in this thesis is done by me under the supervision of Dr. Md. Monirul Islam, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. I also declare that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

Countersigned,

Dr. Md. Monirul Islam
(Supervisor)


Aug 20, 2004

Mohammad Masud Hasan

Acknowledgements

All praises are for the Almighty ALLAH who is the most beneficent and the most merciful.

First of all, I would like to thank my supervisor Dr. Md. Monirul Islam, Assistant Professor, Department of CSE, BUET for teaching me how to carry out a research work. I express my heart-felt and most sincere gratitude to him for his constant supervision, valuable advice and continual encouragement, without which this thesis would have not been possible.

Special thanks should be given to Prof. Dr. Md. Shamsul Alam, Head of the CSE Department, for his continuous suggestions regarding the thesis.

I would like to thank the other members of my examination board Prof. Dr. M. Kaykobad and Prof. Dr. Md. Quamrul Ahsan for their valuable suggestions and of course for their unconditional encouragements.

I must have to acknowledge with sincere thanks the constant support and patience of my family members- my parents specially. The arduous task of having completed the thesis could not have been accomplished without their assistance.

Finally I would like to acknowledge the all-out cooperation and services rendered by the members of the Department of CSE, BUET.

Abstract

This thesis works with a new evolutionary system for feedforward artificial neural networks (ANNs). An indirect encoding scheme, to be particular, modified cellular encoding (MCE) is proposed to represent ANNs. The original cellular encoding is modified in such a way that it does not suffer from the well-known permutation problem or competing conventions problem of genetic algorithms for evolving ANNs. The functionality of some program symbols in cellular encoding is changed; new rules are added. As a consequence, it is possible to apply crossover operator in the genetic search. Radical change of architecture i.e. behaviour from parents to their children is stopped by keeping the application of crossover on genotypes within certain levels. It is shown in this work that addition / deletion of nodes / connections can evidently be done by crossover alone. Other attempts are also taken to minimize behavioural disruption between parents and their offspring. In the evolution system, the number of user specified parameters is also decreased.

The evolutionary system is also implemented and its performance is tested on some real world problems. The upshot of the genetic search is studied and assessed against the contemporary researches, although direct comparison with other evolutionary approaches to designing ANN is very difficult. It is shown in this thesis that the genetic search can find a reasonable ANN from the search space in considerably short period.

Contents

Board of Examiners	i
Declaration	ii
Acknowledgements.....	iii
Abstract	iv
 Chapter 1	
Introduction	
1.1 Introduction.....	1
1.2 Literature Review.....	3
1.3 Objective of this Thesis	8
1.4 Thesis Organization	9
 Chapter 2	
Basic Concepts	
2.1.1 The Analogy to the Human Brain.....	10
2.1.1.1 The Biological Neuron.....	11
2.1.1.2 The Artificial Neuron.....	12
2.1.2 Design	12
2.1.2.1 Layers.....	13
2.1.2.2 Connections.....	14
2.1.2.3 Learning	16
2.1.3 Types of ANN.....	16
2.1.4 Areas of Application	17
2.2 Encoding Scheme.....	20
2.2.1 Direct Encoding	21
2.2.1.1 Connection-based Encoding	21
2.2.1.2 Node-based Encoding	23
2.2.1.3 Layer-based Encoding	24
2.2.1.4 Pathway-based Encoding.....	24
2.2.2 Indirect Encoding.....	24
2.3 Evolutionary Operators.....	25
2.3.1 Population	26
2.3.2 Mutation.....	26
2.3.3 Crossover	26
2.4 Permutation Problem	27
2.5 The Learning Process.....	28
2.5.1 Memorization Paradigms.....	29
2.5.2 Learning Rules.....	29
2.5.2.1 Hebb's Rule	30
2.5.2.2 Hopfield Law	30
2.5.2.3 The Delta Rule.....	30

2.5.2.4 Kohonen's Learning Law	31
2.5.3 Learning Approaches	31
2.5.3.1 Unsupervised Learning	32
2.5.3.2 Supervised Learning	32
2.5.3.3 Off-line or On-line	34
2.6 Selection Mechanisms	34
2.6.1 Roulette Wheel Selection	34
2.6.2 Rank Selection	35
2.6.3 Steady-State Selection	36
2.6.4 Elitism	37
 Chapter 3	
Modified Cellular Encoding	
3.1 Cellular Encoding	38
3.1.1 Basics of CE	38
3.1.2 Developing ANN from CE	40
3.1.3 Properties of CE	44
3.2 Program Symbol Set Used	46
3.3 Modification to Symbol Functionalities	51
3.4 Other MCE Properties	57
 Chapter 4	
The Genetic Search Scheme	
4.1 Evolutionary Approaches	58
4.1.1 Genetic Algorithm	59
4.1.2 Evolutionary Programming	60
4.1.3 Evolution Strategy	61
4.1.4 Genetic Programming	61
4.2 Crossover on MCE	61
4.3 CE to DE Conversion	64
4.4 The Evolutionary System	65
 Chapter 5	
Experimental Studies	
5.1 Data Sets Applied	68
5.1.1 Heart Disease	70
5.1.2 Diabetes	70
5.1.3 Thyroid	71
5.1.4 Breast Cancer	71
5.2 Experimental Setup	72
5.3 Results	73
5.4 Comparison with other works	82
 Chapter 6	
Conclusion & Future Research	
6.1 Concluding Remarks	87

6.2 Future Directions	88
Bibliography	90
Appendix A.....	98
Neural Network Glossary	98

List of Figures

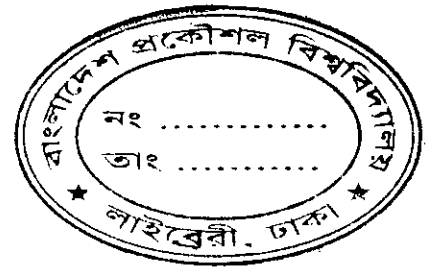
Figure 1.1: The evolution of the most popular artificial neural networks.....	8
Figure 2.1: A simplified biological neuron.....	11
Figure 2.2: An artificial neuron.	12
Figure 2.3: Layers in ANN.	13
Figure 2.4: Different ANN structures.....	19
Figure 2.5: Direct encoding example.....	21
Figure 2.6: An ANN and its genotypic representation.	27
Figure 2.7: The behaviour of a neuron.....	28
Figure 2.8: An illustration of roulette wheel selection.	35
Figure 2.9: An example of rank selection, situation before ranking.....	36
Figure 2.10: An example of rank selection, situation after ranking.....	36
Figure 3.1: Simple Cellular Instance (CI).....	40
Figure 3.2: Interpretation of Simple PST.....	43
Figure 3.3: An example describing Properties 1 and 3.....	54
Figure 3.4: An example describing Properties 2 and 3.....	55
Figure 3.5: Permutations of nodes in an ANN.....	58
Figure 4.1: Problem solution using EA.....	59
Figure 4.2: Major steps of the evolutionary system.....	66
Figure 5.1: Generation Vs Error for diabetes.....	75
Figure 5.2: Generation Vs Connections for diabetes.	75
Figure 5.3: Epoch Vs Error for diabetes.	75
Figure 5.4: Time Vs Error for diabetes.....	76
Figure 5.5: Generation Vs Error for Thyroid.....	76
Figure 5.6: Generation Vs Connections for Thyroid.	77
Figure 5.7: Epoch Vs Error for Thyroid.	77
Figure 5.8: Time Vs Error for Thyroid.	77
Figure 5.9: Generation Vs Error for heart disease.	78
Figure 5.10: Generation Vs Connections for heart disease.....	78
Figure 5.11: Epoch Vs Error for heart disease.....	79
Figure 5.12: Time Vs Error for heart disease.	79
Figure 5.13: Generation Vs Error for breast cancer.....	80
Figure 5.14: Generation Vs Connections for breast cancer.	80
Figure 5.15: Epoch Vs Error for breast cancer.	81
Figure 5.16: Time Vs Error for breast cancer.	81
Figure 5.17: PST of the best ANN found in breast cancer problem.....	82
Figure 5.18: DE of the best ANN found in breast cancer problem.....	82
Figure A.1: A multilayer perceptron.....	101

List of Tables

Table 2.1: Application areas of different artificial neural networks.....	21
Table 2.2: Application areas of different ANNs grouped by network structure.....	22
Table 3.1: Realization of END symbol.....	47
Table 3.2: Realization of PAR symbol.....	48
Table 3.3: Realization of SEQ symbol.....	49
Table 3.4: Realization of CUT / CLIP symbol.....	50
Table 3.5: Realization of MRG symbol.....	51
Table 4.1: Effects of crossover on CE.....	62
Table 5.1: Experimental outcomes.....	74
Table 5.2: Comparison with EPNet for heart disease problem.....	83
Table 5.3: Comparison with EPNet for diabetes problem.....	83
Table 5.4: Comparison with EPNet for Thyroid problem.....	84
Table 5.5: Comparison with EPNet for breast cancer problem.....	84
Table 5.6: Comparison with other works for heart disease problem.....	85
Table 5.7: Comparison with other works for breast cancer problem.....	85
Table 5.8: Comparison with other works for diabetes problem.....	86
Table 5.9: Comparison with other works for thyroid problem.....	86

List of Notations

ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
BP	Back Propagation
CE	Cellular Encoding
DE	Direct Encoding
EP	Evolutionary Programming
ES	Evolutionary Strategy
GA	Genetic Algorithm
GP	Genetic Programming
LMS	Least Mean Square
LVQ	Linear Vector Quantization
MCE	Modified Cellular Encoding
MLP	Multi-layer Perceptron
n	Number of nodes in the output layer
O_{max}	Maximum values of output coefficients
O_{min}	Minimum values of output coefficients
PST	Program Symbol Tree
RSG	Random Selection Group
SOM	Self Organizing Map
T	Total number of input pattern
$Y(i,t)$	Actual output for the i -th output neuron of the t -th input pattern
$Z(i,t)$	Desired output for the i -th output neuron of the t -th input pattern



Chapter 1

Introduction

1.1 Introduction

For decades after Darwin laid down its basic principles, evolution was the domain of biologists and paleontologists. When the synthetic theory brought the successful union of Darwinian principles with Mendelian genetics at the turn of the nineteenth century, most biologists were confident that they had a solid conceptual basis for biology. The mathematical theory of evolution came to be dominated by population genetics, which was commonly thought to provide a sufficiently deep theoretical framework for analyzing the constituent mechanisms driving evolutionary processes. Over the same period that witnessed the flourishing of evolutionary science, starting in the mid- to late-nineteenth century, new concepts and methods were developed in mathematics and the natural sciences that now promise to remove several of the roadblocks to an integrative theory of evolutionary systems.

Evolutionary computation has provided an alternative to the more classical search and optimization methods in recent years. Classical methods tend to get stuck in local optima. One of the advantages of evolutionary computation is that the algorithms do not start from a local search point but explore different areas of the search space in parallel. Other advantages are that they have no presumptions with regard to the search space, that they are widely applicable, that they can be interpreted, that they provide several alternative solutions to the problem at hand and that they are easily combined with other methods.

The idea of using evolutionary computation as a problem solving technique exists since the 1950s. Since then, four major approaches have evolved [25]: Evolutionary Programming (EP), Evolution Strategies (ES), Genetic Algorithms (GA) and Genetic

Programming (GP). All these algorithms have been inspired by the notions of evolution and survival in nature.

Artificial neural networks (ANNs), also referred to as neuromorphic systems, artificial intelligence and parallel distributed processing, are an attempt at mimicking the patterns of the human mind. Many researches have concluded that understanding the human mind is probably the most difficult challenge left in science. Consequently, ANNs have seen an explosion of interest over the last few years, and are being successfully applied across an extraordinary range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, ANNs are being introduced. This sweeping success can be attributed to a few key factors like power, easy of use and applicability.

The power of ANNs is that they are very sophisticated modeling techniques capable of modeling extremely complex functions. In particular, ANNs are nonlinear. For many years linear modeling has been the commonly used technique in most modeling domains since linear models have well-known optimization strategies. Where the linear approximation was not valid (which was frequently the case) the models suffered accordingly. ANNs keep in check the curse of dimensionality problem that bedevils attempts to model nonlinear functions with large numbers of variables.

Another key factor is the ease of use. ANNs learn by example. An ANN user gathers representative data, and then invokes training algorithms to automatically learn the structure of the data. Although the user does need to have some heuristic knowledge of how to select and prepare data, how to select an appropriate ANN, and how to interpret the results, the level of user knowledge needed to successfully apply ANNs is much lower than would be the case using (for example) some more traditional nonlinear statistical methods.

Also, ANNs are applicable in virtually every situation in which a relationship between the predictor variables (independents, inputs) and predicted variables (dependents, outputs) exists, even when that relationship is very complex and not easy to articulate in the usual terms of "correlations" or "differences between groups." The computing world has a lot to gain from ANNs. ANNs also contribute to other areas of research such as neurology and psychology.

1.2 Literature Review

The field of ANNs has a history of some five decades but has found solid application only in the past fifteen years, and the field is still developing rapidly. In the early 1940's scientists came up with the hypothesis that neurons, fundamental, active cells in all animal nervous systems might be regarded as devices for manipulating binary numbers. Thus spawning the use of computers as the traditional replicants of ANNs.

To be understood is that advancement has been slow. Early on it took a lot of computer power and consequently a lot of money to generate a few hundred neurons. In relation to that consider that an ant's nervous system is composed of over 20,000 neurons and furthermore a human being's nervous system is said to consist of over 100 billion neurons! To say the least replication of the human's neural networks seemed daunting. However, today ANNs are being applied to an increasing number of real- world problems of considerable complexity. The history of ANNs that was described above can be divided into several periods [46]:

a) First Attempts: There were some initial simulations using formal logic. During the decade of the first electronic computer, McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "a or b" and "a and b". Another attempt was by using computer simulations. There were two groups (Farley and Clark, 1954; Rochester, Holland, Haibit and Duda, 1956) [42]. The first group (IBM researchers) maintained close contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a multidisciplinary trend which continues to the present day.

b) Promising and Emerging Technology: Not only was neuroscience influential in the development of neural networks, but psychologists and engineers also contributed to the progress of neural network simulations. Rosenblatt (1958) stirred considerable interest

and activity in the field when he designed and developed the Perceptron. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. Another system was the ADALINE (ADaptive LInear Element) which was developed in 1960 by Widrow and Hoff (of Stanford University) [46]. The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron, it employed the Least-Mean-Squares (LMS) learning rule.

c) Period of Frustration and Disrepute: In 1969 Minsky and Papert wrote a book in which they generalized the limitations of single layer Perceptrons to multilayered systems. In the book they said: "...our intuitive judgment that the extension (to multilayer systems) is sterile". The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.

d) Innovation: Although public interest and available funding were minimal, several researchers continued working to develop neuromorphical based computational methods for problems such as pattern recognition. During this period several paradigms were generated which modern work continues to enhance. In 1988, Grossberg's influence founded a school of thought which explores resonating algorithms [42]. They developed the ART (Adaptive Resonance Theory) networks based on biologically plausible models. Anderson and Kohonen developed associative techniques independent of each other. Klopff in 1972 developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis.

In 1974, Werbos developed and used the back-propagation learning method, however several years passed before this approach was popularized. Back-propagation nets are probably the most well known and widely applied of the neural networks today. In essence, the back-propagation net is a Perceptron with multiple layers, a different

threshold function in the artificial neuron, and a more robust and capable learning rule. Amari (A. Shun-Ichi 1967) was involved with theoretical developments: he published a paper which established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification. While Fukushima developed a step wise trained multilayered neural network for interpretation of handwritten characters. The original network was published in 1975 and was called the Cognitron.

f) Re-Emergence: Progress during the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology. Academic programs appeared and courses were introduced at most major Universities (in US and Europe). Attention is now focused on funding levels throughout Europe, Japan and the US and as this funding becomes available, several new commercial with applications in industry and financial institutions are emerging.

A totally unique kind of network model is the Self-Organizing Map (SOM) introduced by Kohonen in 1982. SOM is a certain kind of topological map which organizes itself based on the input patterns that it is trained with. The SOM originated from the LVQ (Learning Vector Quantization) network the underlying idea of which was also Kohonen's in 1972. Hopfield brought out his idea of a neural network in 1982. Unlike the neurons in Multilayered Perceptron (MLP), the Hopfield network consists of only one layer whose neurons are fully connected with each other. Since then, new versions of the Hopfield network have been developed. The Boltzmann machine has been influenced by both the Hopfield network and the MLP. Adaptive Resonance Theory (ART) was first introduced by Carpenter and Grossberg in 1983. The development of ART has continued and resulted in the more advanced ART II and ART III network models.

The application area of the MLP networks remained rather limited until the breakthrough in 1986 when a general backpropagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland.

Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe in 1988. Although the basic idea of RBF was developed 30 years ago under the name method of potential function, the work by Broomhead & Lowe opened a new frontier in the neural network community [42]. The development of ANNs has proceeded as described in Figure 1.

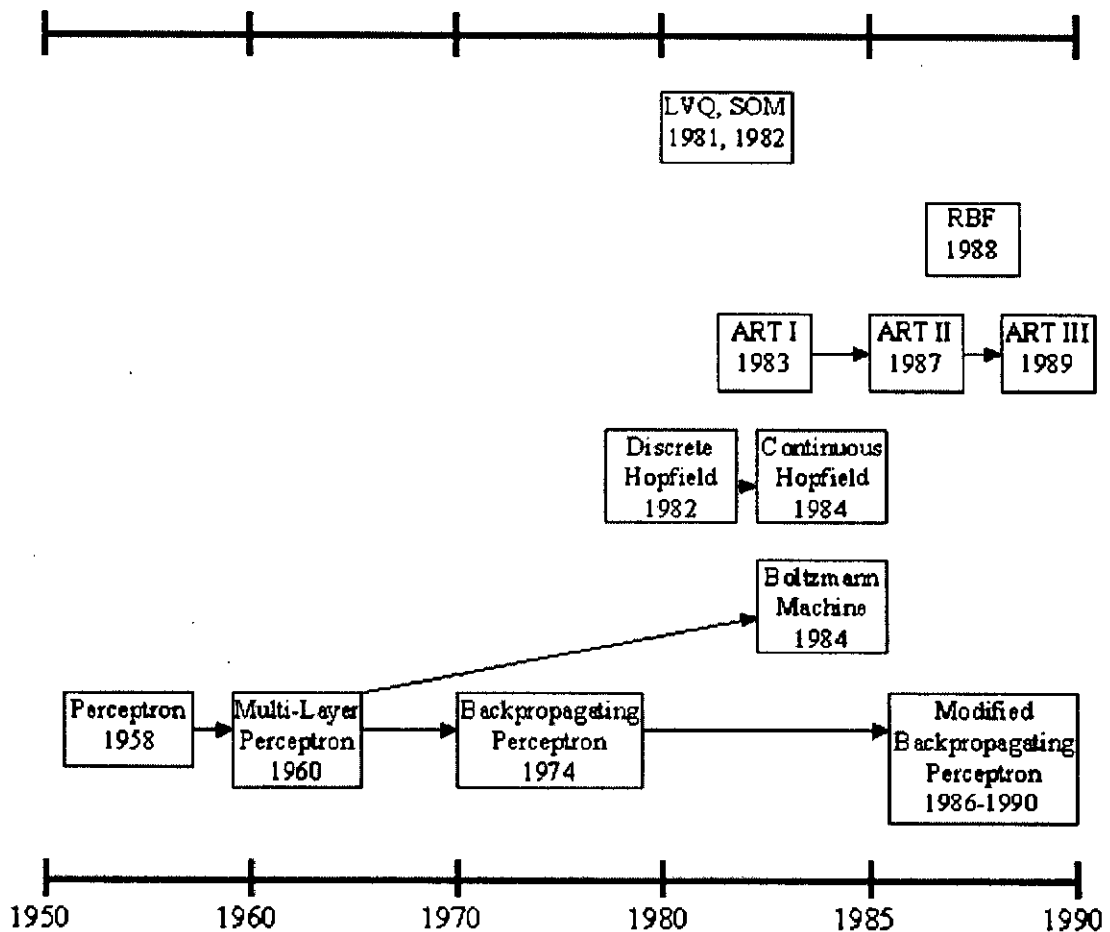


Figure 1.1: The evolution of the most popular artificial neural networks.

g) Today: Significant progress has been made in the field of neural networks—enough to attract a great deal of attention and fund for further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neural based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

Most applications of ANNs use feedforward networks and variants of the classical backpropagation (BP) algorithm. All these training algorithms assume a fixed ANN architecture. They only train weights in the fixed architecture that includes both connectivity and node transfer functions. Many attempts have also been made in designing ANN architectures automatically, such as various constructive and pruning algorithms [29], [38], [48], [49], [50].

Associative memories (AMs) can be implemented using networks with or without feedback. In [63], a two layer feedforward ANN is utilized and proposed a new learning algorithm that efficiently implements the association rule of AM. In order to find an appropriate architecture for a large scale real world application automatically and efficiently, a natural method is to divide the original problem into a set of subproblems. In [51], a simple ANN task decomposition method based on output parallelism is presented. Hsin et. al. [26] suggest divide and conquer learning (DCL) schemes for the design of modular ANNs. When a training process in a multilayer perceptron falls into a local minimum or stalls in a flat region, the proposed DCL scheme is applied to divide the current training data region into two easier to be learned regions. In [37], a constructive algorithm for training cooperative neural network ensembles (CNNEs) is presented which have good generalization ability. Paul et. al. [40] show the use of parallel self scaling quasi-Newton (QN) optimization techniques to improve the rate of convergence of the training process for ANNs.

Angeline et. al. [43] indicate two problems of constructive and pruning hill climbing methods: they may be trapped at local optima and they do not investigate complete class of network architectures. That is why researchers [35], [62] argued on behalf of evolutionary algorithms for finding a near optimal system in the ANN architecture search space.

The central task in evolving ANNs is finding a genetic representation, also called chromosome, genotype or encoding, for an ANN [35]. It dictates how the search landscape is structured, and how scalable the method is [41]. Importance has to be given on the optimal representable structures, excluding meaningless structures, yielding valid offspring by the genetic operators etc. Since the first attempts to combine genetic algorithm and neural network started in the late 1980s, other researchers have joined the

research and created a flood of papers. A variety of different encoding methods does outcome. Two main directions of ANN encoding are direct encoding and indirect encoding. As characterized by Whitely in 1992, low level or direct encoding techniques mostly specify directly parameters such as connectivity or weight values in the genome. Researchers proposed different types of direct encoding based on connections, nodes, layers, pathway etc.

On the other hand, indirect encoding techniques specify not the parameter themselves but production rules that define how to generate these parameters are encoded. This is biologically motivated by the fact that in case of the human brain, there is much more neurons than nucleotides in the genome. So, there has to be a more efficient way of description. Probably, the first indirect encoding scheme was proposed during 1990 by Kitano [23]. Boers and Kuiper [16] proposed another indirect encoding system which was based on Lindemayer's [2] biological model. But may be the most sophisticated encoding method is developed by Frederic Gruau [18] in 1994 in his PhD thesis, which is called cellular encoding. Yet, cellular encoding is not a fully precise representation method. As argued by Talib Hussain [55] in 1997, the scope of improvement defining cellular encoding would be the components of a representation, possible properties of a cell and limit of a cell has on its navigation of the program symbol tree (PST).

In this thesis work, an attempt is taken to find out an enhanced cellular encoding technique so that it can be applied in ANN search space through evolutionary operators.

1.3 Objectives of this Thesis

This thesis work focuses on interactions between ANN's indirect encoding techniques with the evolutionary algorithms. It tries to remove a well known problem of evolutionary neural network encoding called permutation problem and develop a fast evolutionary search scheme with this genome. In summary, the targets are:

- ✓ Representing ANNs using a new indirect encoding scheme i.e. modified CE scheme that does not suffer from permutation problem.
- ✓ Introducing a new evolutionary system for feedforward ANNs.

- ✓ Applying crossover operator in the genetic search with the intention to reduce the number of user specified parameters.
- ✓ Examine the effects of crossover on cellular encoded genotype.
- ✓ Finding out attempts to reduce the noise in fitness evaluation and minimize behavioural disruption between parents and their offspring.
- ✓ Presenting an algorithm to convert genotype from CE to DE.
- ✓ Applications of the new approach with some real world problems and analysis the result.

1.4 Thesis Organization

The organization of the rest of this thesis is as follows: Chapter 2 represents preliminaries of ANN and its evolution. The biological motivation of artificial neural network, its structure, types, encoding and applications, the evolutionary operators, permutation problem, ways of training and selecting good network etc are covered here.

Chapter 3 starts with the basics of the original cellular encoding. It presents how to develop an ANN from the cellular encoding. Then some modifications are suggested and its new properties are described.

Chapter 4 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.

The experimental setup, dataset used and the experimental outcome are given in Chapter 5. An analytical review of the result and the comparison with other works are also given.

Chapter 6 concludes with a summary of the thesis and a few additional remarks about future research directions.

Chapter 2

Basic Concepts

In this chapter preliminaries of neural network and its evolution are presented. At first, in Section 2.1, the biological aspects of artificial neural network, its structure, types and applications are given. Section 2.2 introduces how neural network can be represented. Then, the evolutionary operators are discussed along with the permutation problem which is concerned in this thesis. Ways of training and selecting good network are presented too. This is a non-technical description; thereby it does not go into depth with mathematical formulas, but tries to give a more general understanding. Definitions, which are not included in this chapter, will be introduced later as they are needed.

2.1 Artificial Neural Network

Artificial Neural Network (ANN) is a system loosely modeled on the human brain. The field goes by many names, such as connectionism, parallel distributed processing, neuro-computing, natural intelligent systems, machine learning algorithms, and ANNs. It is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. Each neuron is linked to certain of its neighbors with varying coefficients of connectivity that represent the strengths of these connections. Learning is accomplished by adjusting these strengths to cause the overall network to output appropriate results.

2.1.1 The Analogy to the Human Brain

The most basic components of ANNs are modeled after the structure of the human brain. Some ANN structures are not closely to the brain and some does not have a biological

counterpart in the brain. However, ANNs have a strong similarity to the biological brain and therefore a great deal of the terminology is borrowed from neuroscience.

2.1.1.1 The Biological Neuron

The most basic element of the human brain is a specific type of cell, which provides us with the abilities to remember, think, and apply previous experiences to our every action. These cells are known as neurons, each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them.

All natural neurons have four basic components, which are dendrites, soma, axon, and synapses. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then output the final result. Figure 2.1 shows a simplified biological neuron and the relationship of its four components.

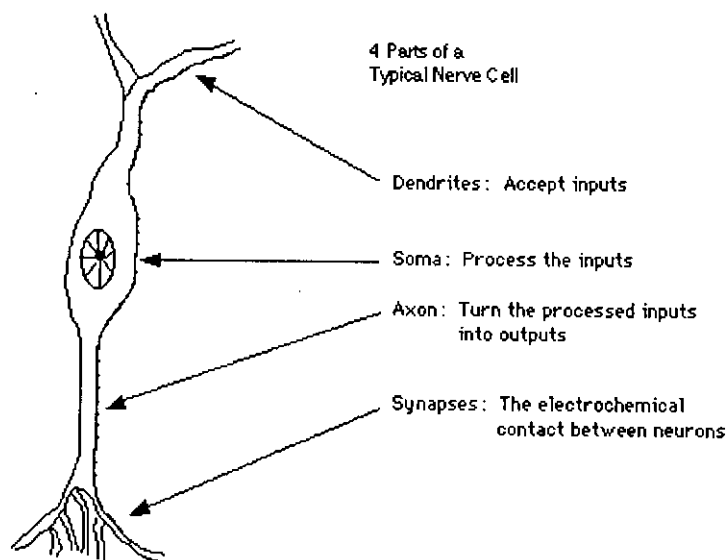


Figure 2.1: A simplified biological neuron.

2.1.1.2 The Artificial Neuron

The basic unit of ANNs, the artificial neurons, simulates the four basic functions of natural neurons. Artificial neurons are much simpler than the biological neuron; the Figure 2.2 below shows the basics of an artificial neuron.

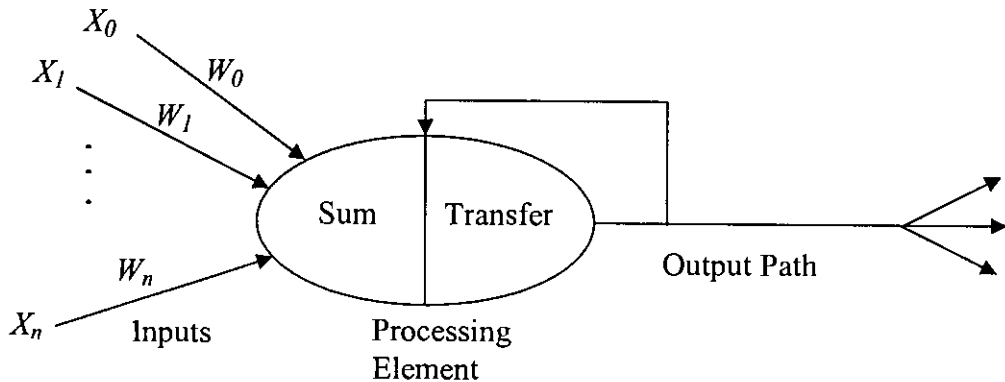


Figure 2.2: An artificial neuron.

Note that various inputs to the network are represented by the mathematical symbol, x_n . Each of these inputs are multiplied by a connection weight, these weights are represented by w_n . In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output. Even though all ANNs are constructed from this basic building block the fundamentals may vary in these building blocks and there are differences.

2.1.2 Design

The developer must go through a period of trial and error in the design decisions before coming up with a satisfactory design. The design issues in ANNs are complex and are the major concerns of system developers.

Designing an artificial neural network consists of:

- Arranging neurons in various layers.

- Deciding the type of connections among neurons for different layers, as well as among the neurons within a layer.
- Deciding the way a neuron receives input and produces output.
- Determining the strength of connection within the network by allowing the network learn the appropriate values of connection weights by using a training data set.
- The process of designing a ANN is an iterative process.

2.1.2.1 Layers

Biologically, ANNs are constructed in a three dimensional way from microscopic components. These neurons seem capable of nearly unrestricted interconnections. This is not true in any man-made network. ANNs are the simple clustering of the primitive artificial neurons. This clustering occurs by creating layers, which are then connected to one another. How these layers connect may also vary. Basically, all ANNs have a similar structure of topology. Some of the neurons interface the real world to receive its inputs and other neurons provide the real world with the network's outputs. All the rest of the neurons are hidden from view.

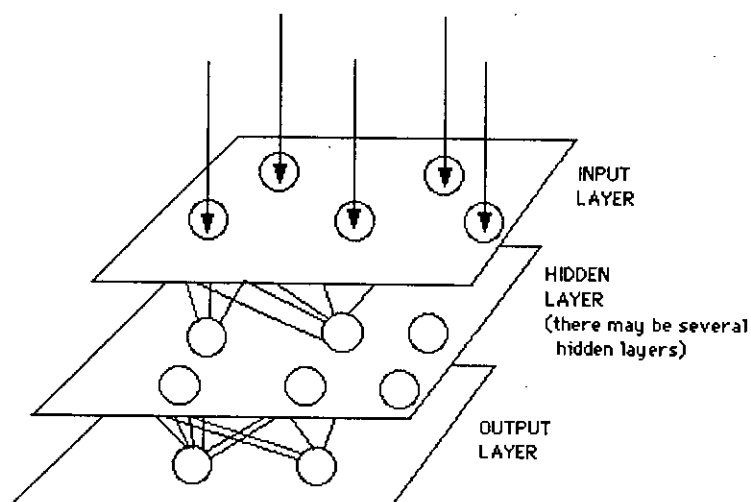


Figure 2.3: Layers in ANN.

As the Figure 2.3 shows, the neurons are grouped into layers. The input layer consists of neurons that receive input from the external environment. The output layer consists of neurons that communicate the output of the system to the user or external environment. There are usually a number of hidden layers between these two layers; the Figure above shows a simple structure with only one hidden layer.

When the input layer receives the input its neurons produce output, which becomes input to the other layers of the system. The process continues until a certain condition is satisfied or until the output layer is invoked and fires their output to the external environment.

To determine the number of hidden neurons the network should have to perform its best, one are often left out to the method trial and error. If you increase the hidden number of neurons too much you will get an over fit, that is the net will have problem to generalize. The training set of data will be memorized, making the network useless on new data sets.

2.1.2.2 Connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there might however be a two-way connection between two neurons, because there may be another path in reverse direction. A neuron receives input from many neurons, but produces a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

There are different types of connections used between layers, these connections between layers are called inter-layer connections.

Fully connected - Each neuron on the first layer is connected to every neuron on the second layer.

Partially connected - A neuron of the first layer does not have to be connected to all neurons on the second layer.

Feed forward - The neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back from the neurons on the second layer.

Bi-directional - There is another set of connections carrying the output of the neurons of the second layer into the neurons of the first layer.

Hierarchical - If an ANN has a hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer.

Resonance - The layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved.

Feed forward and bi-directional connections could be fully or partially connected. In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections.

Recurrent - The neurons within a layer are fully- or partially connected to one another. After these neurons receive input from another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer.

On-center/off surround - A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is allowed to update its and its gang member's weights.

There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.

2.1.2.3 Learning

The brain basically learns from experience. ANNs are sometimes called machine learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights. The learning ability of a ANN is determined by its architecture and by the algorithmic method chosen for training. This topic is described in more details in the section 2.5.

2.1.3 Types of ANN

The ANNs can be classified according to the structure that they exhibit. Figure 2.4 represents four commonly used ANN structures [42].

Figure 2.4 a) represents the structure of a multi-layered feedforward network, the most commonly used. The neurons in this ANN model are grouped in layers which are connected to the direction of the passing signal (from left to right in this case). There are no lateral connections within each layer and also no feedbackward connections within the network. The best-known ANN of this type is the perceptron network.

Figure 2.4 b) depicts a single-layered fully connected network model where each neuron is laterally connected to all neighbouring neurons in the layer. In this ANN model, all neurons are both input and output neurons. The best-known ANN of this type is the Hopfield network.

Figure 2.4 c) demonstrates the connections in a two-layered feedforward / feedbackward network. The layers in this ANN model are connected to both directions. As a pattern is presented to the network, it 'resonate's a certain number of times between the layers before a response is received from the output layer. The best-known ANN of this type is the Adaptive Resonance Theory (ART) network.

Figure 2.4 d) illustrates the idea of a topologically organized feature map. In this model, each neuron in the network contains a so-called feature vector. As a pattern from the training data is given to the network, the neuron whose feature vector is closest to the

input vector is activated. The activated neuron is called the best matching unit (BMU) and it is updated to reflect input vector causing the activation. In the process of updating the BMU, the neighbouring neurons are updated towards the input vector or away from it (according to the learning algorithm in use). The network type exhibiting this kind of behaviour is the Self-Organizing Map of Kohonen.

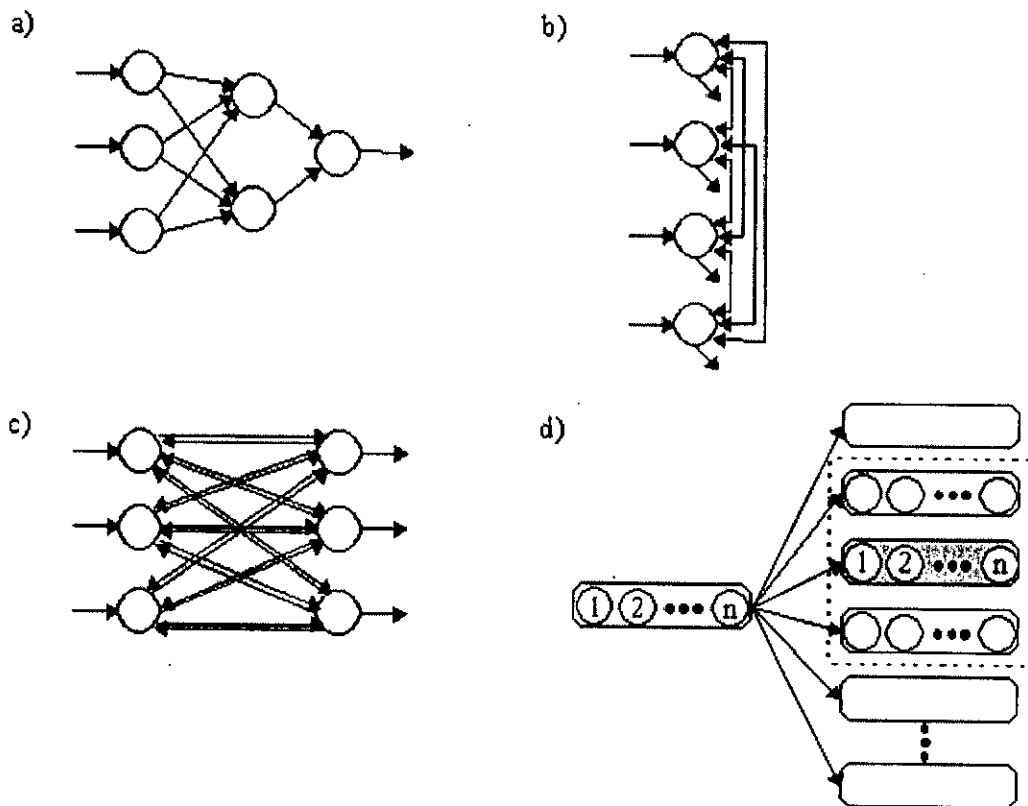


Figure 2.4: Different ANN structures. a) Multi-layered feedforward network, b) single-layered fully connected network, c) two-layered feedforward / feedbackward network and d) topographically organized vector map.

2.1.4 Areas of Application

ANNs are performing successfully where other methods do not, recognizing and matching complicated, vague, or incomplete patterns. ANNs have been applied in solving a wide variety of problems.

The most common use for ANNs is to project what will most likely happen. There are many areas where prediction can help in setting priorities. For example, the emergency

room at a hospital can be a hectic place, to know who needs the most critical help can enable a more successful operation. Basically, all organizations must establish priorities, which govern the allocation of their resources. ANNs have been used as a mechanism of knowledge acquisition for expert system in stock market forecasting with astonishingly accurate results. ANNs have also been used for bankruptcy prediction for credit card institutions.

Although one may apply ANN systems for interpretation, prediction, diagnosis, planning, monitoring, debugging, repair, instruction, and control, the most successful applications of ANNs are in categorization and pattern recognition. Such a system classifies the object under investigation (e.g. an illness, a pattern, a picture, a chemical compound, a word, the financial profile of a customer) as one of numerous possible categories that, in return, may trigger the recommendation of an action such as a treatment plan or a financial plan.

A company called Nestor, have used ANN for financial risk assessment for mortgage insurance decisions, categorizing the risk of loans as good or bad. ANNs has also been applied to convert text to speech, NETtalk is one of the systems developed for this purpose. Image processing and pattern recognition form an important area of ANNs, probably one of the most actively research areas of ANNs.

Another type of research for application of ANNs is character recognition and handwriting recognition. This area has use in banking, credit card processing and other financial services, where reading and correctly recognizing handwriting on documents is of crucial significance. The pattern recognition capability of ANNs has been used to read handwriting in processing checks, the amount must normally be entered into the system by a human. A system that could automate this task would expedite check processing and reduce errors. One such system has been developed by HNC (Hecht-Nielsen Co.) for BankTec.

One of the best known applications is the bomb detector installed in some U.S. airports. This device called SNOOPE, determine the presence of certain compounds from the chemical configurations of their components.

In a document from International Joint conference, one can find reports on using ANNs in areas ranging from robotics, speech, signal processing, vision, character recognition to musical composition, detection of heart malfunction and epilepsy, fish detection and

classification, optimization, and scheduling. One may take under consideration that most of the reported applications are still in research stage.

Basically, most applications of ANNs fall into the following five categories [4], [8], [17]:

- **Prediction** - Uses input values to predict some output. e.g. pick the best stocks in the market, predict weather, identify people with cancer risk.
- **Classification** - Use input values to determine the classification. e.g. is the input the letter A, is the blob of the video data a plane and what kind of plane is it.
- **Data association** - Like classification but it also recognizes data that contains errors. e.g. not only identify the characters that were scanned but identify when the scanner is not working properly.
- **Data conceptualization** - Analyze the inputs so that grouping relationships can be inferred. e.g. extract from a database the names of those most likely to be a particular product.
- **Data filtering** - Smooth an input signal, e.g. take the noise out of a telephone signal.

Table 2.1 illustrates the use of well-known ANNs [42]. Table 2.2 lists the application areas grouped according to the ANN structure.

Table 2.1: Application areas of different artificial neural networks.

Application	Network model			
	Back-propagation	Hopfield	Boltzmann machine	Kohonen SOM
Classification	•	•	•	•
Image processing	•			•
Decision-making	•		•	•
Optimization		•	•	•

Table 2.2: Application areas of different ANNs grouped by network structure.

Structure	Single-layer, lateral connections	Topological vector map	Two-layer feedforward/ feedbackward	Multi-layer, feedforward
Network type	Hopfield	LVQ Kohonen SOM	ART	Perceptron-network Boltzmann machine
Application area	Autoassociation Optimization	Autoassociation Pattern recognition Data compression Optimization	Heteroassociation Pattern recognition	Heteroassociation Pattern recognition Data compression Filtering Optimization

2.2 Encoding Scheme

The genotype representation of a neural architecture is critical to the working of an evolutionary ANN design system. Considerations have to be taken so that the optimal structures are representable in it, meaningless structures are excluded, genetic operators yield valid offspring, and the representation do not grow in proportion to the network. Ideally, the representation should be able to span all potentially useful structures and omit unviable network genotypes. The encoding scheme also constrains the decoding process. For example, a ANN requiring a recurrent structure should have a representation expressive enough to describe recurrent networks. Also the decoding mechanism should be able to read this representation and transform it into an appropriate recurrent network. Since first attempts to combine GA and NN started in the late 1980s, other researchers have joined the movement and created a flood of journal articles, technical reports etc. A variety of different encoding strategies have been implemented. This section tries to structure this information. It does not emphasize on the complete review of single approaches, it rather attempted to gather more general information.

Two main directions of ANN encoding are direct encoding and indirect encoding. Low level or direct encoding techniques mostly specify the connections only. Indirect encodings are more like grammatical rules; these rules suggest a context free graph grammar according to which the network can be generated. Direct encoded genotypes increase very fast in length with a growing network. Thus, the maximum topological

space has to be limited by the user. This may exclude the fittest structure in the lot, or may result in networks with special connectivity patterns.

2.2.1 Direct Encoding

In this thesis, the term “direct encoding” [14] refers to encoding strategies that directly encode parameters of the neural net such as weight values, connection information, etc. into the genome. This is opposed to “indirect encoding”, where rules or alike are encoded which carry information how the network has to be constructed. An example is shown in the Figure 2.5.

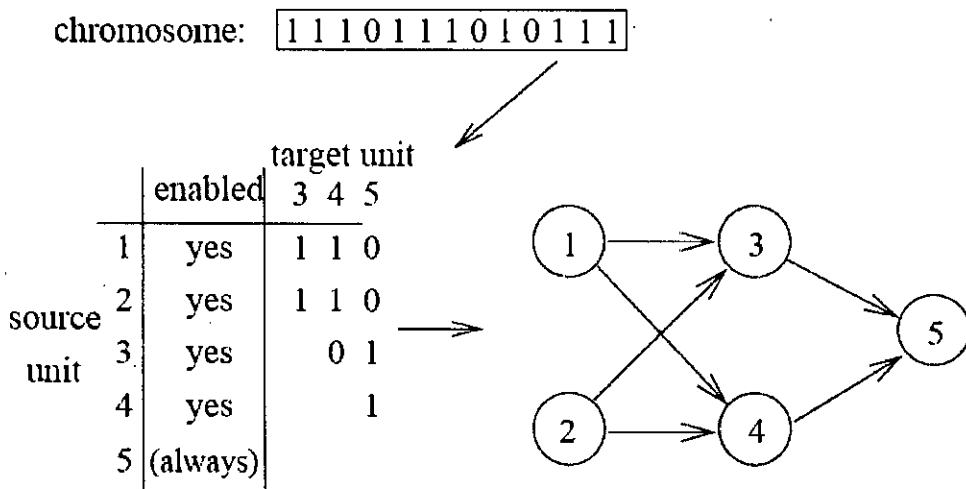


Figure 2.5: Direct encoding example

2.2.1.1 Connection-based Encoding

Most of the encoding strategies that can be encountered in literature are connection-based. This means, that they encode and optimize the connectivity of a network architecture that is predefined.

a) Innervator

In [22], a GANN system called innervator is described that searches for effective architectures for simple functions such as XOR. Based on a five-node feed-forward network model, each connection is encoded by a bit. For fitness evaluation, each generated

individual is trained over a certain number of epochs. Fast convergences toward architectures that are able to solve the task are reported.

b) Genitor

Maybe the most influential approach is Whitley's GENITOR [15]. A lot of researchers copied or reinvented his encoding strategy in order to, for instance, evaluate the quality of the GANN idea (such as [24]).

c) Real-Valued Encoding

An encoding strategy similar to GENITOR was proposed in [11], [13]. It encodes the weights as real number, thus the genome is a list of real numbers. A couple of adapted mutation operators are used that apply to the changed format. Results in of the successful application of this encoding strategy to classification tasks are reported.

d) Variable Length of Weight Encoding

An interesting improvement of Whitley's GENITOR is proposed by Maniezzo [56], [57]. He is adding the evolution of granularity to the original approach. He argues that the binary encoded weights result in a bit-string that is too long for efficient evolution of the genetic algorithm. Thus, he encodes the number of bits per weight in the parameter string. This enables the algorithm to first efficiently find a proper topology with a small number of bits. At a later stage, the fine tuning of the individuals occurs with an increasing number of bits per weight.

e) Shuffle of Encoding Positions of Connections

In [31], an "outer" genetic algorithm is proposed. Its purpose is to optimize the placement of the connection encodings on the genome. Experimental results show that the performance of the standard weight-based encoding scheme can be improved by an advanced placement.

2.2.1.2 Node-based Encoding

The class of connection-based encoding strategies has the disadvantage that basic (maximal) network architecture has to be designed. As mentioned before, however, the choice of the number of neurons is difficult to make. It would be an advantage, if the user of a GANN system would be released from this decision. A solution is promised by node-based encoding strategies. As indicated by the name, the parameter string does no longer consist of weight values, but entire node information.

a) Schiffmann

A quite simple example of node-based encoding was developed by Schiffmann, Joost and Werner [58], [59], [60]. The parameter string is a list of nodes including connectivity information. In an early version called “BP-Generator”, this connectivity information included weight values. The later version is reduced to mere existence information of a connection.

b) GANNet

A node-based encoding for layered networks called GANNet is proposed by [9]. Its basic structure is quite similar to the previous one. The parameter string is a list of neurons with connectivity and placement information. However, the architecture of the neural net is more restricted.

c) Koza

Koza applies his genetic programming paradigm to ANNs by choosing a node-based encoding strategy [27]. The nodes are not encoded in a parameter string, but rather in a parameter tree.

d) Related Strategies

Related to these kinds of node-based encoding are contributions such as “GA-delta” [46], where only one layer of neurons is optimized via a genetic algorithm. A very simple

encoding is proposed by Bishop and Bushnell, who merely encode the number of neurons per layer for a color recipe prediction task [28].

2.2.1.3 Layer-based Encoding

The first layer -based encoding scheme was proposed in [53], [54]. The genome consists of a certain number of areas, each of which encodes one layer of the network. The nodes of each layer are arranged in three dimensions. Each area contains a layer ID, information about the number and arrangement of nodes, and a varying number of projector fields that contain connectivity information. A variation of this encoding scheme is presented in [36], which is based on layers.

2.2.1.4 Pathway-based Encoding

A different approach is taken in [7]. Here, not connections, nodes or layers are encoded, but pathways through the network. A path is defined as a list of neurons beginning with an input neuron and ending with an output neuron. There is no further restriction to the order of nodes in the paths. Hence, the strategy does not necessarily define feed-forward networks.

2.2.2 Indirect Encoding

The encoding strategies mentioned in section 2.2.1 share a common property: They directly encoded parameter such as connectivity or weight values in the genome. Opposed to that, this section presents some indirect encoding strategies, which is also called “recipe” encoding in [60]. Here, not the parameter themselves, but production rules, that define how to generate these parameters are encoded. This is biologically motivated by the fact, that in case of the human brain, there is much more neurons than nucleotides in the genome - for instance [5]. So, there has to be a more efficient way of description. As pointed out in [16], the organization of a biological organism shows a great deal of modularity. Its division into cells and the division of the brain into neurons

exemplifies this. The skin is composed of the same kind of skin cells all over the body. Encoding strategies that use rules to copy units with the same function can be more efficient in producing large networks.

a) Grammar Encoding Method

Probably the first indirect encoding scheme was proposed by Kitano [23], also called grammar encoding method. The encoded rules rewrite an initial weight matrix, whose structure is similar to the one used in Innervator. However, the entries are non-terminal symbols of the grammar.

b) Cellular Encoding

Maybe the most sophisticated encoding method is developed by Frederic Gruau [18], which is called cellular encoding. The genome encodes an set of instructions that are applied to an initial network, consisting of one hidden node. During the execution of the instructions, a larger network evolves.

These instructions cover operations such as duplication of a node, deletion of a connection, sequential division and a couple of more complex operations that include the use of registers. Recursive calls allow the modular constructions of large networks with a relative small instruction set. This scheme will be discussed more details in chapter 3.

c) Lindenmayer-Systems

Lindenmayer-Systems are based on a biological model proposed by Aristid Lindemayer [2]. It tries to simulate the cellular development of organisms, where cells only exchange information with their neighbors without central control. Boers and Kuiper [16] describe how this model can be used to encode ANNs.

2.3 Evolutionary Operators

This section gives mainly the idea about some basic genetic operators such as crossover and mutation.

2.3.1 Population

Where most classical optimization methods maintain a single best solution found so far, an evolutionary algorithm maintains a population of candidate solutions. Only one (or a few, with equivalent objectives) of these is "best," but the other members of the population are "sample points" in other regions of the search space, where a better solution may later be found. The use of a population of solutions helps the evolutionary algorithm avoid becoming "trapped" at a local optimum, when an even better optimum may be found outside the vicinity of the current solution.

2.3.2 Mutation

Inspired by the role of mutation of an organism's DNA in natural evolution -- an evolutionary algorithm periodically makes random changes or mutations in one or more members of the current population, yielding a new candidate solution (which may be better or worse than existing population members). There are many possible ways to perform a "mutation," and the Evolutionary Solver actually employs three different mutation strategies. The result of a mutation may be an infeasible solution, and the Evolutionary Solver attempts to "repair" such a solution to make it feasible; this is sometimes, but not always, successful.

2.3.3 Crossover

Inspired by the role of sexual reproduction in the evolution of living things - an evolutionary algorithm attempts to combine elements of existing solutions in order to create a new solution, with some of the features of each "parent." The elements (e.g. decision variable values) of existing solutions are combined in a "crossover" operation, inspired by the crossover of DNA strands that occurs in reproduction of biological organisms. As with mutation, there are many possible ways to perform a crossover operation -- some much better than others -- and the Evolutionary Solver actually employs multiple variations of two different crossover strategies.

2.4 Permutation Problem

The evolution of ANN architectures in general suffers from the permutation problem [44], [47] (also called competing conventions problem [14], [44] or structural-functional mapping problem [15]). It is caused by the many to one mapping from genotypes to phenotypes. In general, it is happened when any permutation of the hidden nodes produces behaviourally equivalent ANN but with different genotypic representation. This problem not only makes the evolution inefficient, but also makes crossover operators more difficult to produce quality descendants [41]. It is unclear what building blocks actually are in this situation. For example, ANNs shown in Figure 2.6(a) and Figure 2.6(c) are equivalent, but they have different genotypic representations as shown by Figure 2.6(b) and Figure 2.6(d) using a direct encoding scheme, assuming that each weight is represented by four binary bits. Zero weight implies no connection. In general, any permutation of the hidden nodes will produce behaviorally equivalent ANN but with different genotypic representations. This is also true for indirect encoding schemes. In order to avoid the detrimental effect of the permutation problem the EPNet [62] algorithm does not use crossover. Hence, the permutation problem in the encodings of ANNs needed to be resolved.

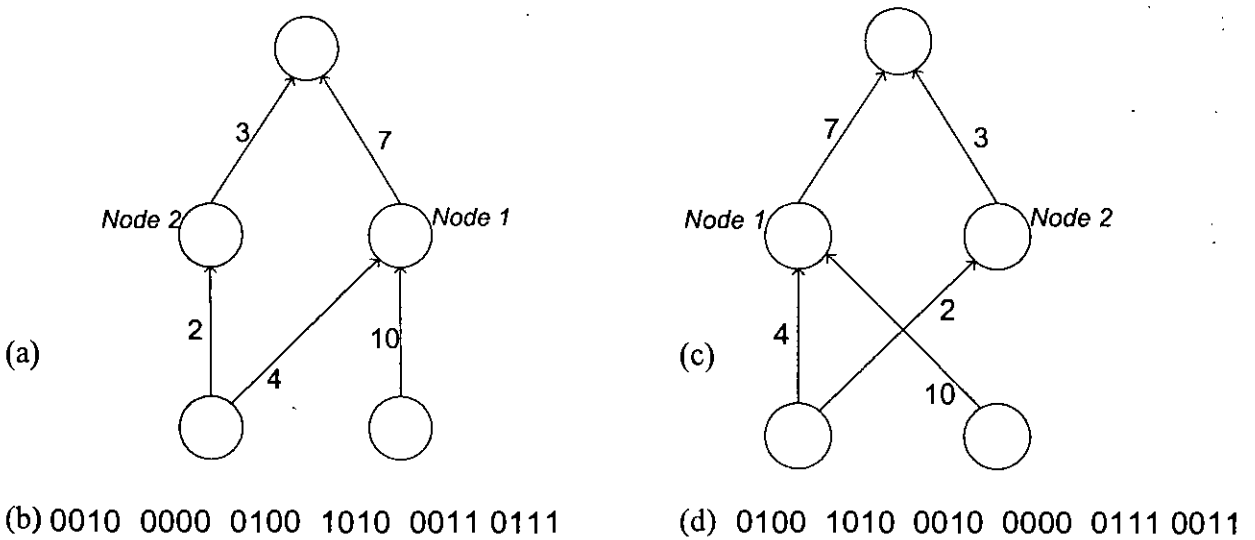


Figure 2.6: (a) An ANN and (b) its genotypic representation, 2.6(c) An ANN which is equivalent to that given in Figure 2.6(a) and (d) its genotypic representation.

2.5 The Learning Process

The brain basically learns from experience. ANNs are sometimes called machine learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights.

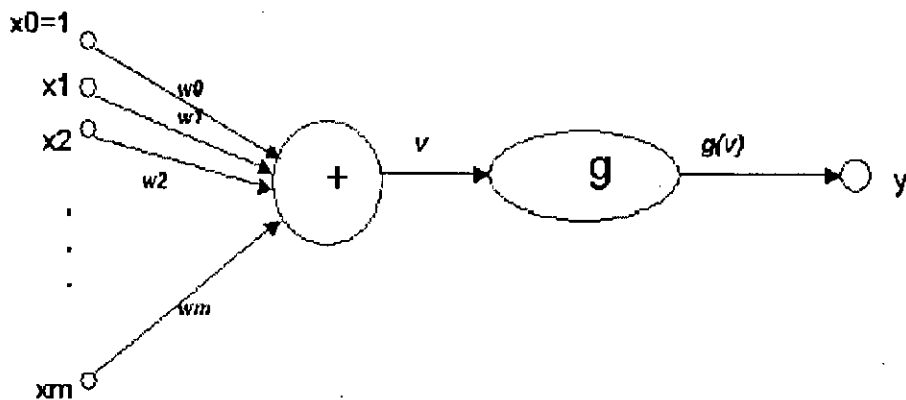


Figure 2.7: The behaviour of a neuron.

The behaviour of an ANN (Artificial Neural Network) depends on both the weights and the input-output function (transfer function), as shown in the Figure 2.7, that is specified for the neurons. This function typically falls into one of three categories:

- linear (or ramp)
- threshold
- sigmoid

For linear units, the output activity is proportional to the total weighted output. For threshold unit, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value. For sigmoid units, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

2.5.1 Memorization Paradigms

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms:

a) Associative mapping in which the network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms:

Auto-association: an input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern completion, i.e. to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns.

Hetero-association: is related to two recall mechanisms: nearest-neighbour and interpolative recalls. In nearest-neighbour recall the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and in interpolative recall the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping, is classification, i.e. when there is a fixed set of categories into which the input patterns are to be classified.

b) Regularity detection in which units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular 'meaning'. This type of learning mechanism is essential for feature discovery and knowledge representation.

2.5.2 Learning Rules

There are a variety of learning rules which are in common use. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule. Man's understanding of how neural processing actually works is very limited. Learning is

certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc. A few of the major laws are given as an example below.

2.5.2.1 Hebb's Rule

The first and the best known learning rule was introduced by Donald Hebb. The description appeared in his book "The organization of Behavior" in 1949. This basic rule is: If a neuron receives an input from another neuron and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened.

2.5.2.2 Hopfield Law

This law is similar to Hebb's Rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate." (Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one.)

2.5.2.3 The Delta Rule

The Delta Rule is a further variation of Hebb's Rule, and it is one of the most commonly used. This rule is based on the idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a neuron. This rule changes the connection weights in the way that minimizes the mean squared error of the network. The error is back propagated into previous layers one layer at a time. The process of back-propagating the network errors

continues until the first layer is reached. The network type called Feed forward, Back-propagation derives its name from this method of computing the error term.

This rule is also referred to as the Windrow-Hoff Learning Rule and the Least Mean Square Learning Rule.

2.5.2.4 Kohonen's Learning Law

This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the neurons compete for the opportunity to learn, or to update their weights. The processing neuron with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted output, and only the winner plus its neighbors are allowed to update their connection weights. The Kohonen rule does not require desired output. Therefore it is implemented in the unsupervised methods of learning. Kohonen has used this rule combined with the on-center/off-surround intra-layer connection to create the self-organizing ANN, which has an unsupervised learning method.

2.5.3 Learning Approaches

Information is stored in the weight matrix of a ANN. Learning is the determination of the weights. Following the way learning is performed, one can distinguish two major categories of ANNs:

Fixed networks in which the weights cannot be changed. In such networks, the weights are fixed a priori according to the problem to solve.

Adaptive networks which are able to change their weights.

The learning ability of a ANN is determined by its architecture and by the algorithmic method chosen for training. All training methods used for adaptive ANNs can be classified into two major categories: supervised learning and unsupervised learning.

2.5.3.1 Unsupervised Learning

Unsupervised learning method is not given any target value. A desired output of the network is unknown. The hidden neurons must find a way to organize themselves without help from the outside. During training the network performs some kind of data compression such as dimensionality reduction or clustering. The network learns the distribution of patterns and makes a classification of that pattern where, similar patterns are assigned to the same output cluster. Kohonen network is the best example of unsupervised learning network. According to Sarle [61], Kohonen network refers to three types of networks that are Vector Quantization, Self-Organizing Map and Learning Vector Quantization.

2.5.3.2 Supervised Learning

In supervised learning, the network user assembles a set of training data. The training data contains examples of inputs together with the corresponding outputs, and the network learns to infer the relationship between the two. Training data is usually taken from historical records. Data are used to adjust the network's weights and thresholds so as to minimize the error in its predictions on the training set. If the network is properly trained, it has then learned to model the (unknown) function that relates the input variables to the output variables, and can subsequently be used to make predictions where the output is not known.

a) Reinforcement Learning

This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged, then reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results.

Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the proper connection weights.

b) Back Propagation

One of the most commonly used supervised NN model is backpropagation network that uses backpropagation learning algorithm. It has been popularized by Rumelhart, Hinton, and Williams in 1980s as a euphemism for generalized delta rule. Backpropagation of errors or generalized delta rule is a decent method to minimize the total squared error of the output computed by the net [32].

In order to train an ANN to perform some task, one must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the ANN compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. That is why sigmoid function is, in general, used as the activation function:

$$\text{Sigmoid}(x) = 1 / (1 + e^{(-x)})$$

The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, first it needs identify all the weights between that hidden unit and the output units to which it is connected. Then it is multiplied those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, one can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection.

Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the EA must be converted into the EI, the rate at which the error changes as the total input received by a unit is changed.

2.5.3.3 Off-line or On-line

One can categorize the learning methods into yet another group, off-line or on-line. When the system uses input data to change its weights to learn the domain knowledge, the system could be in training mode or learning mode. When the system is being used as a decision aid to make recommendations, it is in the operation mode, this is also sometimes called recall.

a) Off-line: In the off-line learning methods, once the system enters into the operation mode, its weights are fixed and do not change any more. Most of the networks are of the off-line learning type.

b) On-line: In on-line or real time learning, when the system is in operating mode (recall), it continues to learn while being used as a decision tool. This type of learning has a more complex design structure.

2.6 Selection Mechanisms

Inspired by the role of natural selection in evolution -- an evolutionary algorithm performs a selection process in which the "most fit" members of the population survive, and the "least fit" members are eliminated. In a constrained optimization problem, the notion of "fitness" depends partly on whether a solution is feasible (i.e. whether it satisfies all of the constraints), and partly on its objective function value. The selection process is the step that guides the evolutionary algorithm towards ever-better solutions. There are many methods in selecting the best offspring. Examples are roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others. Some of them will be described here.

2.6.1 Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes (genotypes of the ANNs) are, the more chances to be selected they have. Imagine a roulette wheel

where all the chromosomes in the population are placed. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value is, the larger the section is. The following picture in Figure 2.8 is given for an example.

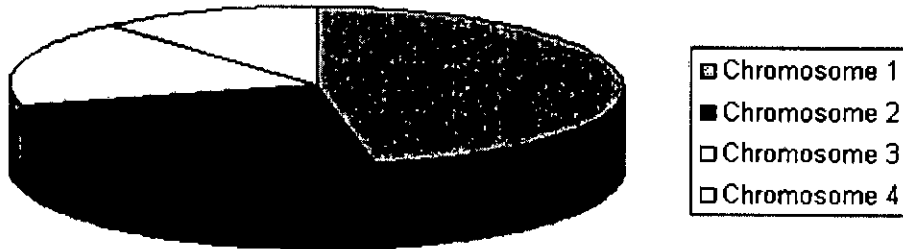


Figure 2.8: An illustration of roulette wheel selection.

A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value will be selected more times. This process can be described by the following algorithm.

[Sum] Calculate the sum of all chromosome fitnesses in population - sum S .

[Select] Generate random number from the interval $(0, S)$ - r .

[Loop] Go through the population and sum the fitnesses from 0 - sum s . When the sum s is greater than r , stop and return the chromosome where we are.

Of course, the step 1 is performed only once for each population.

2.6.2 Rank Selection

The previous type of selection will have problems when they are big differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses then the other chromosomes will have very few chances to be selected.

Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population).

The following pictures in Figure 2.9 and Figure 2.10 show how the situation changes after changing fitness to the numbers determined by the ranking.

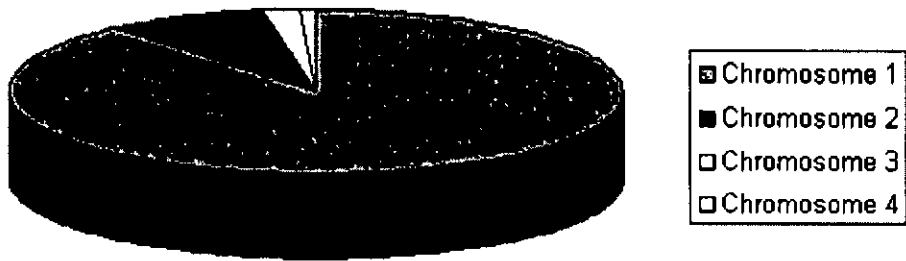


Figure 2.9: An example of rank selection, situation before ranking (graph of fitnesses).

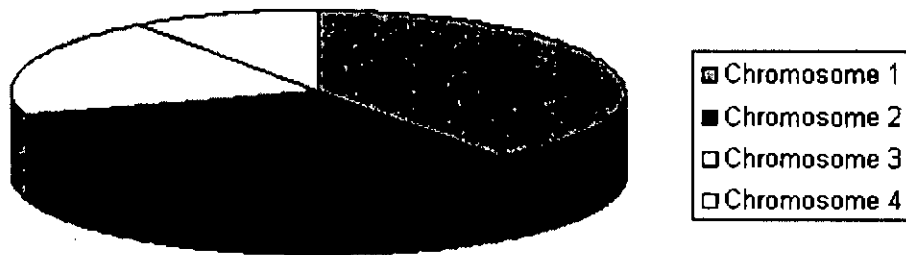


Figure 2.10: An example of rank selection, situation after ranking (graph of order numbers).

Now all the chromosomes have a chance to be selected. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

2.6.3 Steady-State Selection

This is not a particular method of selecting parents. The main idea of this type of selecting to the new population is that a big part of chromosomes can survive to next generation.

The steady-state selection GA works in the following way. In every generation a few good (with higher fitness) chromosomes are selected for creating new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

2.6.4 Elitism

When creating a new population by crossover and mutation, one has a big chance, that one will lose the best chromosome. To prevent this elitism is used. It first copies the best chromosome (or few best chromosomes) to the new population. The rest of the population is constructed in ways described above. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best found solution.

Chapter 3

Modified Cellular Encoding

In this chapter fundamental aspects and stages of developing ANN from the cellular encoding, in the context of the original cellular encoding system, is discussed first. Then some modifications are suggested. The new system is called modified cellular encoding.

3.1 Cellular Encoding

An important direction in NN research is the development of systematic methods for the exploration of this space of possibilities. Even more important is the development of automatic forms of such systematic methods. Gruau [18] has proposed an initial step towards such automatic exploration by defining a system capable of generating a wide variety of ANN's from representations formed from a small class of operators. Gruau has named his method Cellular Encoding (CE) and the basics of his approach shall be presented here.

3.1.1 Basics of CE

In CE, a particular artificial neural network (ANN) is specified through the application of a sequence of graph transformations to an initial graph. The transformations operate upon graphs which may have two classes of nodes, either cells or neurons. These two nodes differ in that each cell has associated with it a program symbol tree (PST) that specifies how that cell will ultimately be replaced by neurons. Each cell of the starting graph (of which there is usually only one) has a PST associated with it. Neurons are the nodes

which make up the ANN that ultimately results from the process. In the end, all cells must have been transformed into neurons. Thus, there is an analogy to phrase structure grammars in that cells are nonterminal symbols in the representation and neurons are terminal symbols. Similarly, the initial graph usually consists of a single cell, from which sentential forms containing both types of nodes are created, and finally only the terminal symbols (i.e., neurons) remain. These forms will thus be referred to as the starting graph (SG), sentential graphs (IG), and the terminal graph (TG).

The analogy to the use of phrase structure grammars can be extended to the idea that the expansion of each cell (non-terminal) into neurons (terminals) is context-free, in that its expansion is independent of its neighbouring nodes in the graph. Thus, with respect to the resultant set of neurons, the expansion of any sentential forms is independent of the order in which the cells are expanded, and in fact, all cells in any sentential form may therefore be expanded in parallel to result in a unique set of neurons. However, the final graph also consists of connections among neurons. The final configuration of connections among neurons may be dependent on the order of expansion of the cells and can therefore be viewed as a context-sensitive component of the ANN generates in this way. The PST associated with a particular cell consists of program symbols that each specifies a particular transformation applicable to that cell. These transformations may replace, expand, duplicate or remove the cell itself within the sentential graph, they may modify the way in which the cell is connected within the sentential graph, and they may modify the cell's internal variables.

To summarize visually, Figure 3.1 shows a simple cellular instance [55] with the starting graph consisting of a single cell a , whose reading head r_a indicates a starting execution at the root of its PST. The labels SEQ and END refer to program symbols which shall be discussed later. In this case, as is required to ultimately obtain a functional ANN, the single cell in the starting graph has initial connections to inputs and outputs.

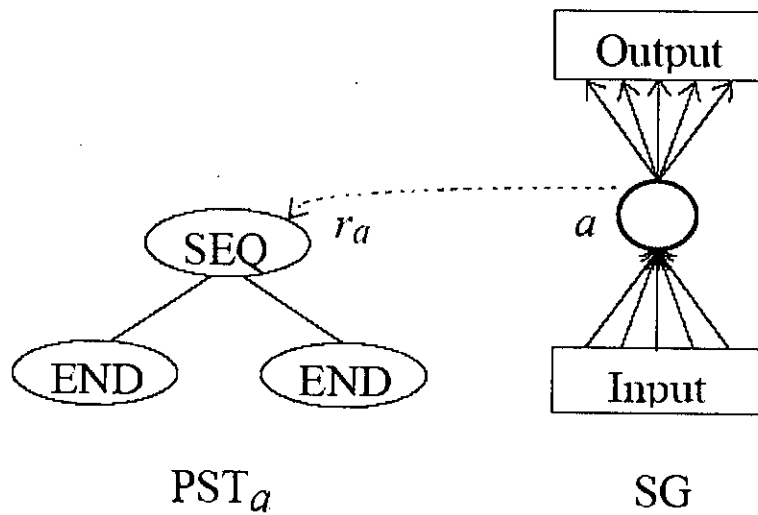


Figure 3.1: Simple Cellular Instance (CI)

3.1.2 Developing ANN from CE

The cellular code is represented as a grammar tree with ordered branches whose nodes are labeled with name of program symbols. The reader must not make the confusion between grammar tree and tree grammar. Grammar tree means a grammar encoded as a tree, whereas tree grammar means a grammar that rewrites trees. A cell is a node of an oriented network graph with ordered connections. Each cell carries a duplicate copy of the cellular code (i.e., the grammar tree) and has an internal reading head that reads from the grammar tree. Typically, each cell reads from the grammar tree at a different position. The character symbols represent instructions for cell development that act on the cell or on connections of the cell. During a step of the development process, a cell executes the instruction referenced by the symbol it reads, and moves its reading head down in the tree. One can draw an analogy between a cell and a Turing machine. The cell reads from a tree instead of a tape and the cell is capable of duplicating itself; but both execute instructions by moving the reading head in a manner dictated by the symbol that is read. Here, the grammar tree is referred as a program and each character as a program-symbol. A program symbol (PS) applied to a cell has four defining functions. Firstly, it has a certain succession function. The PS must specify whether and how that cell is to be superseded in the sentential graph. The possible succession actions are to remain in the

IG, expand through the creation of new successor cells which are placed in the IG, eliminate itself from the IG, or to terminate and become a neuron in the IG. Any one or more of these operations may be performed by a PS, although in the interest of simple, atomic program symbols, usually only one or two are performed. In Gruau, for example, several program symbols eliminate the cell and replace it with two successor cells.

Secondly, a PS has a certain structure function. The PS must specify what changes, if any, are to be made to the connectivity of the cell within the sentential graph and to the neuronal variables of the cell. If the cell has successors, the PS must specify their connectivity and neuronal variables as well. For example, a program symbol may have a single connection of the cell removed. One important issue concerning the structure function is the scope of the changes to connectivity that is permitted. How connections are stored and changed is an implementation issue, but there is the question of where they are ideally considered to be stored - i.e., locally in the cell or globally as a shared variable of the sentential graph - and how they may be changed - i.e., may a cell connect to any other cell in the sentential graph or are there limitations. The decision about this at the ideal level determines the consequences of the order of execution of the cells. In Gruau, the connections are stored locally, but a cell is able to make changes to other cells and to connect to any cell by maintaining a complete connection matrix of all cells.

Thirdly, a PS has a certain completion function. The PS must specify what action is to be performed by the cell upon the completion of its succession and structure functions. In particular, the PS must specify which program symbol in the cell's PST is to be executed next. To specify the execution of another program symbol, the PS must use and modify the cell's developmental variables. In particular, the reading head of the cell must be set to point to a different node in the PST. If the cell has successors, the PS must specify what their first action should be. Of course, if the cell was eliminated from the sentential graph or transformed into a terminal neuron, then it has no next action. The consequence of such a completion function is that the program symbols in a cell's PST are not processed in a standard order, such as depth-first or breadth-first. Rather, the order is determined dynamically by the resetting of the reading head that accompanies the execution of each PS. In some cases, the resulting order can be quite haphazard. In Gruau, most PSs simply move the reading head to a child in the tree, but some move the

reading head to the root or to arbitrary positions in the tree. Finally, a PS has a certain priority function. In addition to the ordering of the execution of program symbols within the PST of a particular cell, there is the more global issue of the order in which the cells themselves have their PSTs executed. In Gruau's work, the cells are not truly independent, and the connectivity in the final graph is often dependent upon the order in which all the program symbols in all the PSTs of all the cells throughout development are executed. To resolve a unique interpretation of the initial graph, each PS must specify what priority should be assigned to the next action of the cell. For example, a PS may assign a low priority to the action, resulting in the cell giving up processor control and allowing another cell to execute first. The consequence of the priority function is that the ordering of the program symbols is determined dynamically during development. Gruau uses a strict priority form with a global first-in, first-out (FIFO) queue structure. At a given time, a number of cells may be on the queue; if the starting graph has one cell, then it is the only entry in the queue at the beginning of development. The cell at the head of the queue has the highest priority and executes its program symbol before the others. Within the execution of a PS, changes may be made to the queue. In Gruau's work, a cell may remove itself from the head, may place itself on the tail, and may place its successors on the tail. Gruau imposes a strict ordering by forcing every PS to always remove the cell from the head of the queue and place it at the tail (i.e., always assigns lowest priority to the next action), and to always place the left successor on the tail before the right successor. Gruau's purpose in using such a structure to determine order of execution is to mimic as closely as possible a parallel execution of the cells, as well as a breadth-first-like traversal of the PST.

The development of the simple starting graph from Figure 3.1 is traced using the cellular encoding processes of Gruau as shown in Figure 3.2. That cell has a reading head which points to the root of the program symbol tree, and that cell can be considered to be the first and only entry in a global FIFO queue. The first step in the development of the graph is to examine the first cell in the FIFO queue. This is done, and it is noted that the cell (a) has a reading head (r_a) pointing to the root of its PST. The root is read, and it is noted that the operation SEQ is written there. The succession function of SEQ is to replace the single ancestor cell, a , with two successor cells, b and c . Each, of course, has its own

copy of the PST of cell a . The structure function of SEQ is to assign the first successor cell, b , the same input connections as a and a single output connection to the second successor cell, c , and to assign c with a single input connection from b and the same output connections as a . What so far found is an intermediate ANN structure consisting of two cells connected in a certain way, and no neurons.

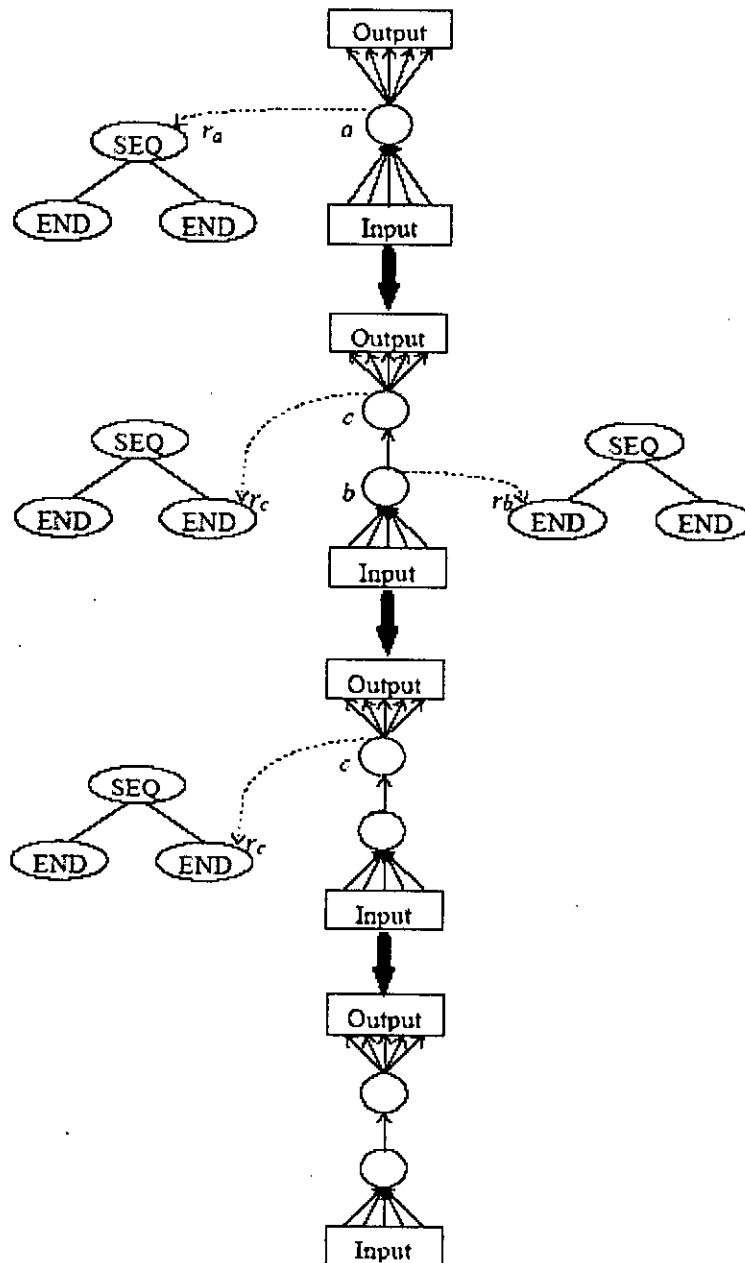


Figure 3.2: Interpretation of Simple PST

The completion function of SEQ is to set the reading head of the first successor, r_b , on the left subtree of the SEQ symbol and the reading head of the second successor, r_c , on the right subtree. The priority function of SEQ is a set of FIFO queue instructions. The instructions, in order, are to remove the original cell from the queue, place the first successor cell onto the FIFO queue, place the second successor on to the FIFO queue, and signal completion. The last will invoke the process of reading the next cell at the head of the queue and starting its development. In this particular case, the next cell is the the first successor, b . One can imagine that if all cells follow similar tree navigation pattern, then the entire process can be imagined as performing a breadth-first, left to right traversal of a single PST. To complete the description of the PST interpretation process, discussion is needed what happens when a terminal program symbol is read. The primary terminal symbol in Gruau's work is END. When a cell labeled with END is read, the succession function is to cease being a cell and replace the cell with no new cells. The structure function is to create a ANN node with the same connection characteristics (e.g., connectivity pattern, specific weight values) as the cell and with certain internal parameters (e.g., activation function, initial bias threshold value, learning rate). For example, after cell b executes its END symbol, an ANN with one neuron (i.e., light circle) with (tentative) connections to the remaining cell, c is found. There is no completion or priority function since the cell has ceased to exist.

3.1.3 Properties of CE

The properties of CE can be summarized as follows:

- **Completeness:** Any ANN can be encoded, thus the GA can reach the solution ANN if it exists at all.
- **Compactness:** The CE is topologically more compact than classical representations of neural net-works, and functionally more compact than any other representation. This ensures that the codes manipulated by the GA are minimal in size. The shorter the codes are, the shorter the search space is, and the less effort the GA has to do.

- **Closure:** The CE always develops meaningful architectures. It produces acyclic or recurrent architectures, depending on the initial graph. Both of these closure properties ensure that the chromosomes produced by the GA always give interesting ANNs. The GA does not throw away any codes. This increases efficiency.
- **Modularity:** If a network can be decomposed into copies of subnetworks, the code of network is the concatenation of the subnetworks code, and a code that describes how to connect the subnetworks. This facilitates the formation of building blocks. The resulting regularities in the final architecture make it clear and interpretable. The code is more compact.
- **Scalability:** A fixed size code encodes a family of ANNs. For any reasonable family of problems, there exists a code such that the i -th ANN solves the i -th problem. A parameterized problem of arbitrary size, say 1000000, can be as easy to solve as the same problem of size 2 or 3 because the code of the corresponding neural net is the same, only the size parameter changes.
- **Power of expression:** The CE can be seen as an "ANN machine language". Its power of expression is greater than Turing machines and cellular automata. It is another proof of the extreme compactness of the coding.
- **Abstraction:** It is possible to compile a program into the code of a ANN that simulates the computation of the program. The wide search space of ANN is transformed into a smaller search space of programs. It helps the GA to find ANNs for problems that can be solved with a short program.
- **Grammars:** Cellular encoding encodes a graph grammar, this property is in fact the root of all the other properties.

According to [55], it seems as though cellular encoding is not a precise representation method. A proper approach to defining cellular encoding would be to:

1. Clearly define the components of a representation.
2. Define the possible properties of a cell.
3. Specify what limits a cell has on its navigation of the PST.

4. Strictly delineate what other functions may and may not be performed during the interpretation of a cell

Once this is done, the basic properties of the encoding approach may be proved.

3.2 Program Symbol Set Used

In this thesis, initially program symbol set {ACYC, END, PAR, SEQ, CUT, WAIT, INCLR, MRG} which is closed and complete is chosen. There are other sets of symbols that also hold such features, but this one is simpler and minimum in cardinality [18]. Among the above program symbols, link register is used only for CUT and MRG symbols. It means that INCLR (and DECLR, [55]) is used with CUT and MRG symbols. If parameters are used with these two symbols to determine the link to be processed, it is not needed INCLR any more.

Again, the symbol WAIT is a skip operation. It is introduced since in some cases the order of cell execution should be varied to be able to represent all ANNs, i.e. for more compact codes. As mentioned in [55], the benefits and drawbacks of WAIT were not analyzed. CE needs WAIT because of Gruau's implementation of connections. He implements them as a matrix, and when a node is added, the matrix is resized to include a new row and new column, and Gruau is very unclear as to how these rows and columns are initialized. Moreover, while it is attempted to get rid of permutation problem from the encoding scheme, WAIT enables operations, which are independent of it, to be reordered. The possibilities of shuffling program symbols with WAIT in a PST permit different encodings for the same ANN. That is why WAIT is not used here directly. The functionality of WAIT is used implicitly whenever needed. For example, a MRG operation that causes deletion of a cell will be delayed until the reading head of the cell is END. Such implicit use of WAIT's functionality not only make sure to remain the completeness and closure properties defined by Gruau but also purge the permutation problem associated with WAIT symbol.

Hence the five program symbols used for CE without recurrent link are END, PAR, SEQ, CUT, and MRG. This modification of CE scheme does not weigh down compactness

property; the closure & completeness properties also hold as well. The following tables (Table 3.1 to 3.5) show their functionalities in brief.

Table 3.1: Realization of END symbol.


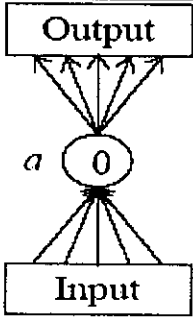
Succession function	Cease being a cell (0 arity)
Structure function	Create ANN node with same connectivity and internal properties as original cell. (A final node is represented as a light circle)
Completion function	Lose reading head
FIFO queue instruction	Signal completion
Visual description	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  </div> <div style="text-align: center;">  </div> </div> <p style="text-align: center;">PST IG</p>

Table 3.2: Realization of PAR symbol.

Succession function	Replace cell with two child cells. (2 arity)
Structure function	Both cells share the same input and output connections as original.
Completion function	(1) Place reading head of first child cell (<i>b</i>) on left subtree (2) Place reading head of second child cell (<i>c</i>) on right subtree
FIFO queue instruction	(1) Place first child cell (<i>b</i>) on queue (2) Place second child cell (<i>c</i>) on queue (3) Signal completion
Visual description	

Table 3.3: Realization of SEQ symbol.

Succession function	Replace cell with two child cells. (2 arity)
Structure function	Create one cell sharing the same input connections as original, one cell sharing the same output connections as original, and a connection from the first to the second.
Completion function	(1) Place reading head of first child cell (<i>b</i>) on left subtree (2) Place reading head of second child cell (<i>c</i>) on right subtree
FIFO queue instruction	(1) Place first child cell (<i>b</i>) on queue (2) Place second child cell (<i>c</i>) on queue (3) Signal completion
Visual description	<p>PST</p> <p>IG</p>

Table 3.4: Realization of CUT / CLIP symbol.

Succession function	The connection indicated by the 'current link' pointer is removed (1 arity)
Structure function	Removal of connection
Completion function	Place reading head of the cell on the unique subtree
FIFO queue instruction	(1) Place the cell on queue (2) Signal completion
Visual description	<p>The diagram illustrates the visual description of the CUT / CLIP symbol. On the left, labeled 'PST', there is an oval labeled 'CLIP' connected by a solid line to another oval labeled 'C'. A dashed arrow labeled r_a points from the 'CLIP' oval to the 'C' oval. On the right, labeled 'IG', there is a box labeled 'Input' at the bottom, connected by three lines to a central circle labeled $a(0,2)$. This circle is then connected by three lines to a box labeled 'Output' at the top.</p>

Table 3.5: Realization of MRG symbol.

Succession function	Given an integer argument i and cell b , let l be the input link which number is i and c the neighbor cell which is connected through l . The program-symbol replaces l by the list of input links of c ,... If after this operation, c has no more output links, c is suppressed. (arity 1)
Structure function	Change connectivity as above and possibly eliminate a node.
Completion function	Place reading head of child cell on unique subtree
FIFO queue instruction	(1) Place child cell on queue (2) (Remove cell c from queue if it is there) (3) Signal completion
Visual description	<p>PST</p> <p>Before</p> <p>After</p>

3.3 Modification to Symbol Functionalities

To solve the permutation problem, first it is done modification of some properties of the program symbols of CE. To understand clearly, let us observe the program symbols more details. Permutation problem arises whenever different orientations of nodes in hidden layers are possible for behaviourally same ANN. Two children created by program symbol SEQ do not have this property since here the output of one child is the input to another child. This link will totally reverse if these two children are arranged in another orientation while creation and hence will get ANN of different phenotype. Program

symbol END has no arity and CUT, MRG have single arity [55]. Therefore permutation problem cannot be generated from their children. But the program symbol PAR is of double arity and its two children grow in parallel having same input-output links. If the two children are arranged in different order different genotypic representations for the same phenotype will be found.

This problem arises from the poor definition of the execution of PAR symbol while interpreting PST (i.e. CE of ANN). According to [18], after execution of PAR the reading head of the first child cell will be placed on left subtree and the reading head of the second child cell will be placed on right subtree. If the reading head of the first child cell is placed on right subtree and the reading head of the second child cell on left subtree (which is another CE), different ANN of same behaviour will be found.

Hence, the interpretation of PAR symbol is redefined so that the decision of placing reading head of a child, either at the left subtree or at the right subtree, will be taken randomly at the time of execution of PAR symbol while interpreting the PST (addressed by Property 3 given below). This ensures that the two children of PAR have equal probability to place reading head at the left (or right) subtree. Consequently, in the PST, any order of two subtrees (hence constructing different redundant CEs) below PAR symbol can produce same set of behaviourally equivalent ANNs. If it is allowed both orders, many to one mapping from genotypes to phenotypes will be found. Instead, it is permitted to appear the program symbols under the PAR symbol in the PST according to a fixed order only (addressed by Property 1 given below). Any order here may suffice (Property 3 explains why it is true), one can select program symbols randomly from the order given in Property 1. This restriction just removes redundancy in the search space by not allowing the occurrences of multiple CEs for the same phenotype.

Property 1: The two children (program symbols) of the PAR symbol in PST can appear according to the following order only, given in descending priority, PAR, SEQ, CUT, MRG, END.

When the two children of PAR in PST are of same type symbol, there is no problem until they are both PAR again. For the case of two PAR children of a parent PAR, another restriction is formulated.

Property 2: At most one child of a PAR symbol in the PST can be a PAR symbol again. According to Property 1, it should be the left child if there is one.

In fact, Property 3 enables us to represent a PAR symbol with two PAR children by a series of PAR symbols in PST. Hence Property 2 does not restrict search space at all; rather it removes redundancy from the search space by stopping to encode same phenotypic ANN in different ways.

Before going to Property 3, first should be introduced a new term Random Selection Group (RSG). If a PAR symbol in the PST has two non-PAR symbols (symbols other than PAR), they (including their corresponding subtrees) constitute the RSG of that PAR symbol. If one child of the PAR symbol is PAR again (it should be left child by Property 1), then the two non-PAR children (along with corresponding subtrees) of this child PAR symbol and the right child (along with corresponding subtree) of the parent PAR symbol constitute the RSG for that parent PAR symbol. This definition of RSG goes recursively.

Property 3: During the interpretations of the children of a particular PAR symbol in PST, one can choose randomly and exclusively any symbol (including corresponding subtree) from the RSG of that particular PAR symbol for all the positions of non-PAR symbols children.

Here exclusively means that if it is chosen one element from RSG for a position this form will be excluded from the RSG and not consider any more. All elements have equal probability to be chosen.

An important point to note is while representing CE with RSGs, one will keep symbols in a RSG according to the order given in Property 1. That is, highest priority symbol of a RSG will appear in the highest level of positions that constitute the RSG. If multiple CUT (or MRG) symbols appear, they also are ordered based on the parameters.

Figure 3.3 describes the interpretations of Properties 1 and 3. ANNs shown in Figure 3.3(a) and 3.3(b) are of same phenotype [62]. CE shown in Figure 3.3(c) can generate both ANNs. Here two children of PAR, CUT 1 (along with its subtree) and END constitute RSG of the parent PAR. According to Property 3, while interpreting the PST one can choose for the left child of PAR anyone from RSG. If one chooses CUT 1 for the left child execution one gets ANN as in Figure 3.3(a). If one chooses END for the left child execution one gets ANN as in Figure 3.3(b). CE shown in Figure 3.3(d) is invalid. According to Property 1 CUT has higher priority than END, hence CUT should be left child and END should be the right child of PAR as shown in Figure 3.3(c).

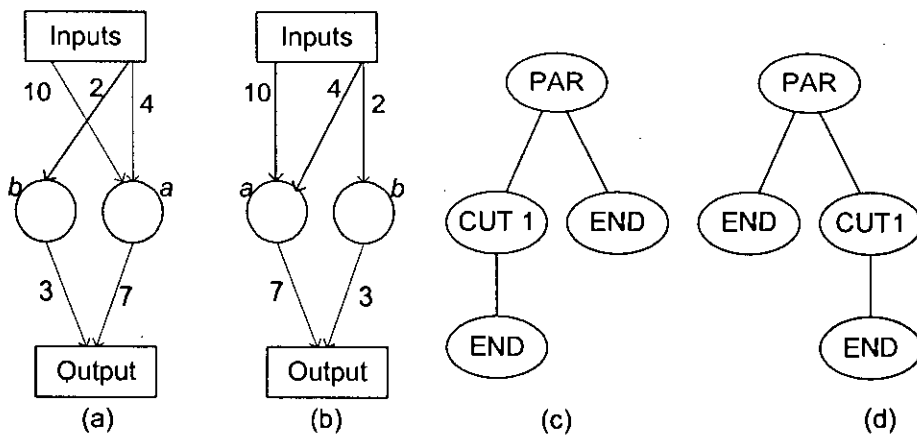


Figure 3.3: An example describing Properties 1 and 3. (a), (b) ANNs of same phenotype. (c) CE of the architecture of both (a) and (b). (d) an invalid encoding of the ANNs.

Figure 3.4 explains Properties 2 and 3. Figure 3.4(a) is an ANN that can be encoded in several ways. Among these only the CE shown in Figure 3.4(b) is valid according to Property 2. Figure 3.4(c) is an invalid representation since it has two PAR children of the parent PAR symbol. Any permutation of the four nodes ($P(4, 4) = 4! = 24$ ways) of the ANN in Figure 3.4(a) will produce behaviourally same ANNs. Figure 3.4(b) can generate any of these ANNs. The RSG of the root PAR symbol constitute of CUT 1, CUT 3, along with their corresponding subtrees, and the two ENDS of the last PAR symbols. One has 4 ways to select from RSG for the first right child. For the second right child one has the remaining 3 ways. For the other two children one has 2 and 1 ways. So, total ways to

execute the PST is $4 \times 3 \times 2 \times 1 = 24$. The general proof is shown in the following Proposition / Theorem.

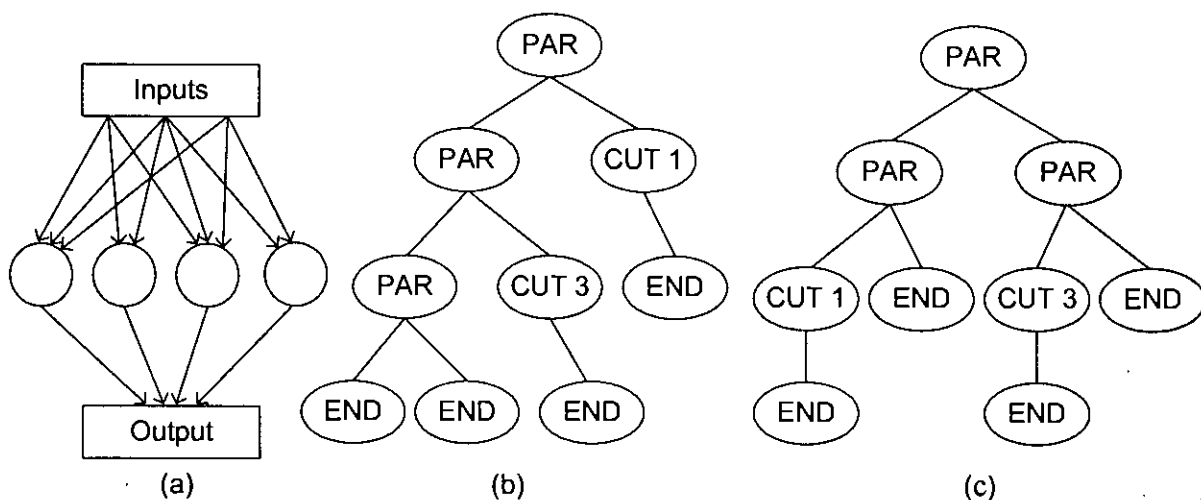


Figure 3.4: An example describing Properties 2 and 3. (a) a given ANN. (b) its valid CE. (c) an invalid encoding.

Proposition 1: Property 1, Property 2, and Property 3 allow a unique CE for all ANNs of same phenotype, thus removing permutation problem.

Proof: Since program symbols SEQ, CUT, MRG and END do not contribute in the permutation problem, it needs to concentrate on PAR symbol only. As a result of the modification of the behaviours of the original PAR symbol the only CE found is shown in Figure 3.5(b) for the ANN as in Figure 3.5 (a). No other organization of PAR symbols to produce this ANN is valid by the Properties 1 and 2.

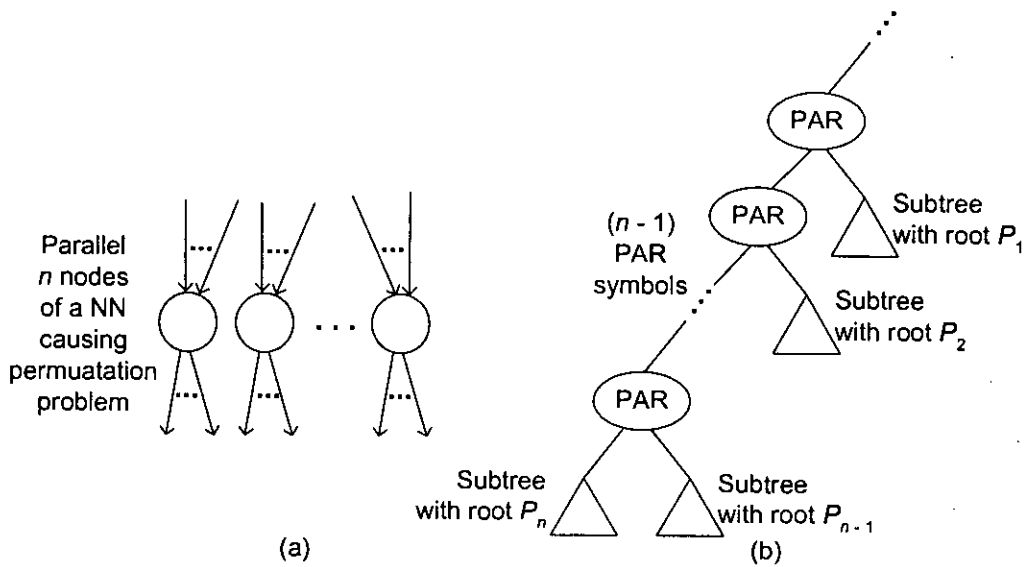


Figure 3.5: Permutations of nodes in an ANN. (a) part of a ANN shows the nodes that may have any permutations. (b) CE of that part of ANN.

The n nodes in Figure 3.5(a) can be oriented in any permutation each gives an ANN with the same behaviour. Since these nodes can be arranged in $P(n, n) = n!$ ways, one has $n!$ ANNs all having same phenotype.

Now, the Random Selection Group (RSG) of the top PAR symbol in Figure 3.5(b) is made of the non-PAR symbols that are the roots of the subtrees shown in triangles. Subtree i is rooted with a program symbol P_i which is not a PAR symbol. Hence, RSG constitutes of $P_1, P_2, P_3, \dots, P_n$ (along with corresponding subtrees). According to Property 3, while executing the first right child one can choose any of the elements from RSG in n ways. For the next right child the remaining $n - 1$ element is left from RSG leaving $n - 1$ ways to choose with. Thus continuing up to the last child it is left with 1 way for the last child. So, there is a total $n \times (n - 1) \times (n - 2) \times \dots \times 1 = n!$ ways to select. It means that the CE of Figure 3.5(b) can represent all $n!$ ANNs that are of same phenotype. Thus, permutation problem is resolved by the modified CE scheme \square

3.4 Other MCE Properties

This subsection summarizes other properties of MCE.

1. When there is only one input link on a given cell, the CUT operation is not applicable for that cell [18], [55]. CUT cannot also be applied on a link when this is the only output link from the parent cell (or from input). Because if it is allowed, the parent cell should be deleted after the operation developing a new PST that can be generated in another (simpler) way arising permutation problem.
2. MRG symbol cannot be applied on the first level of the ANN i.e. on the cells that are directly connected to the inputs. Like CUT, when there is only one input link on a given cell, the MRG operation is not applicable for that cell [18], [55].
3. The parameters of CUT and MRG can be used as modulus $n + 1$ where n is the number of current input links of the cell now considering. If so, one needs not worried or be informed about the number of current input links when one set the parameter.
4. First symbol (root) of PST must be PAR, SEQ or END.

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.

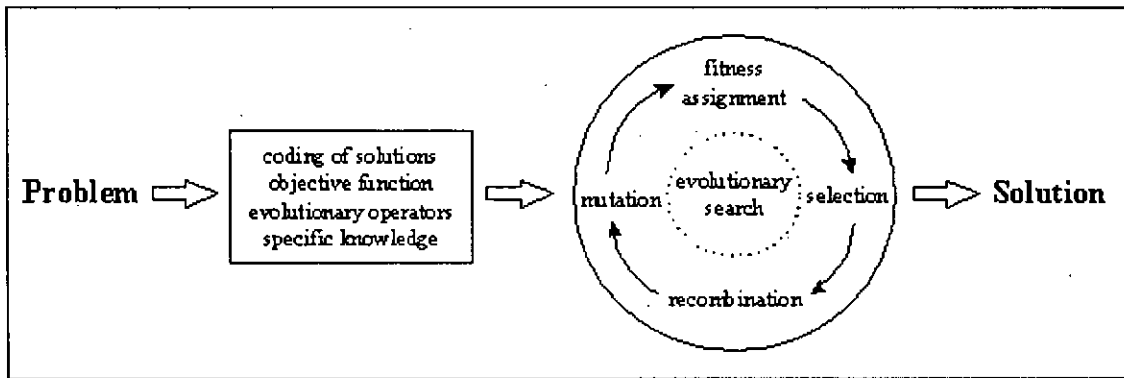


Figure 4.1: Problem solution using EA.

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.

<i>Effects of Crossover</i>		<i>Replaced by</i>				
		PAR	SEQ	CUT	MRG	END
<i>To be replaced</i>	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 \equiv no change in the architecture of corresponding ANNs

NA \equiv node addition

ND \equiv node deletion

CA \equiv connection addition

CD \equiv connection deletion

OC \equiv new orientation of connection

OCN \equiv 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** \neq 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.

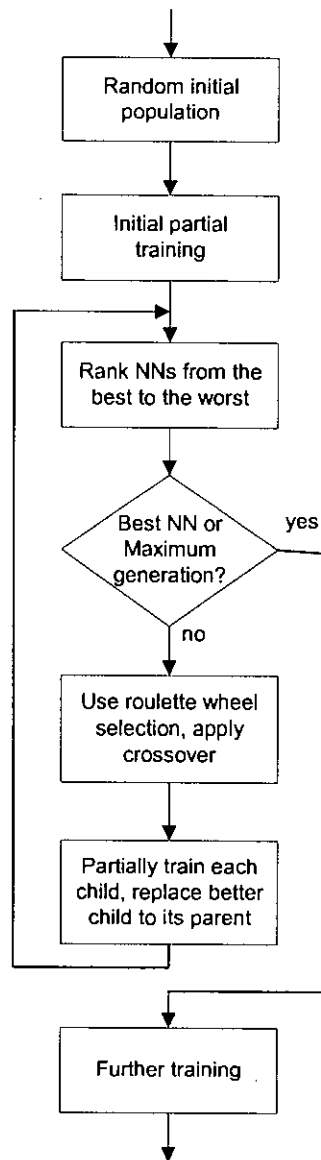


Figure 4.2: Major steps of the evolutionary system.

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](ftp://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 Randolph Wernero obtained from Daimler-Benz. Two files were provided. “antrain.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

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

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 \times generation \times ANN) is $200 \times 1 \times 20 = 4000$ 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.

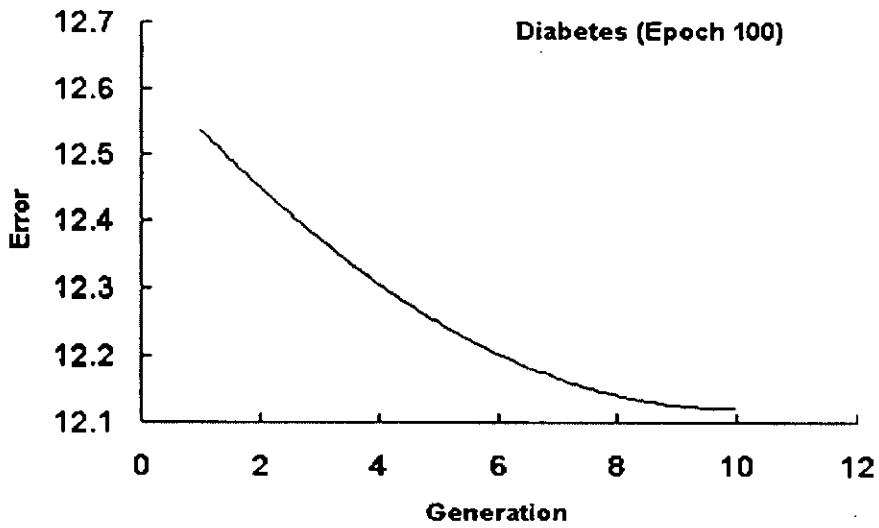


Figure 5.1: Generation Vs Error for diabetes.

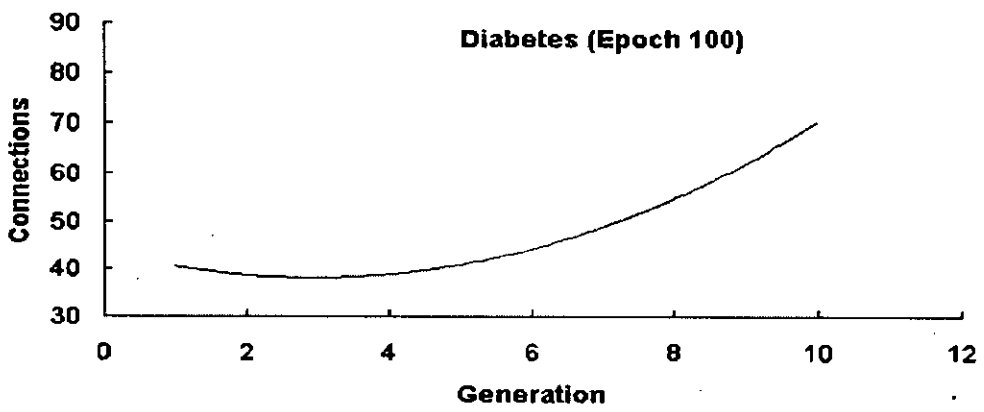


Figure 5.2: Generation Vs Connections for diabetes.

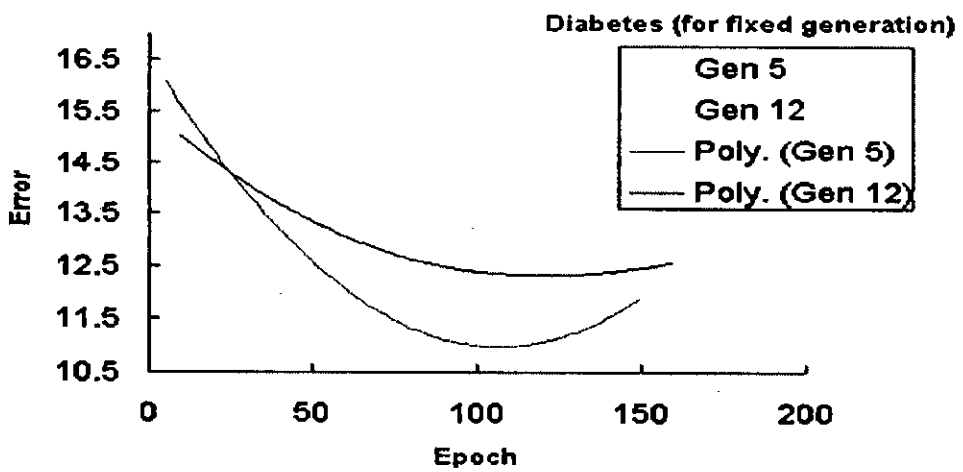


Figure 5.3: Epoch Vs Error for diabetes.

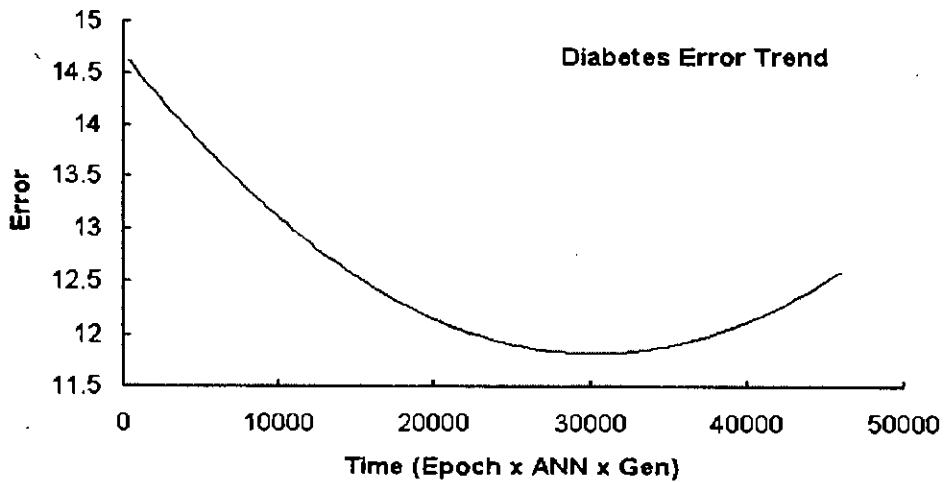


Figure 5.4: Time Vs Error for diabetes.

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.

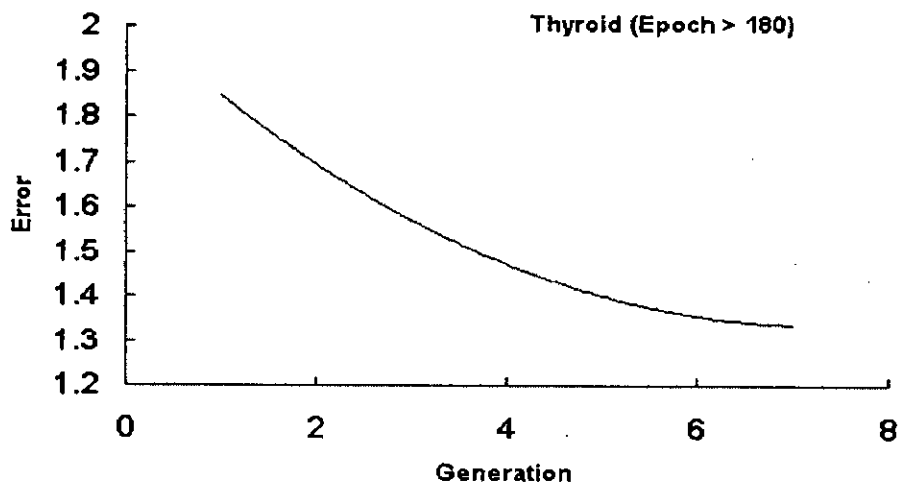


Figure 5.5: Generation Vs Error for Thyroid.

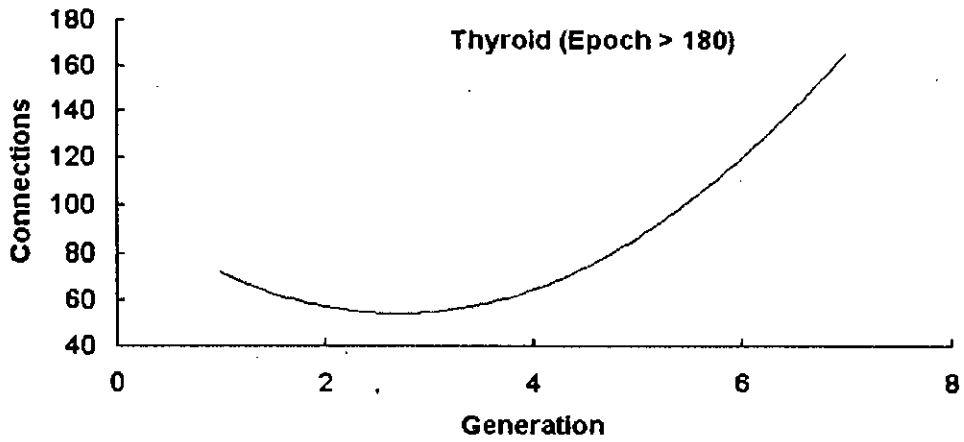


Figure 5.6: Generation Vs Connections for Thyroid.

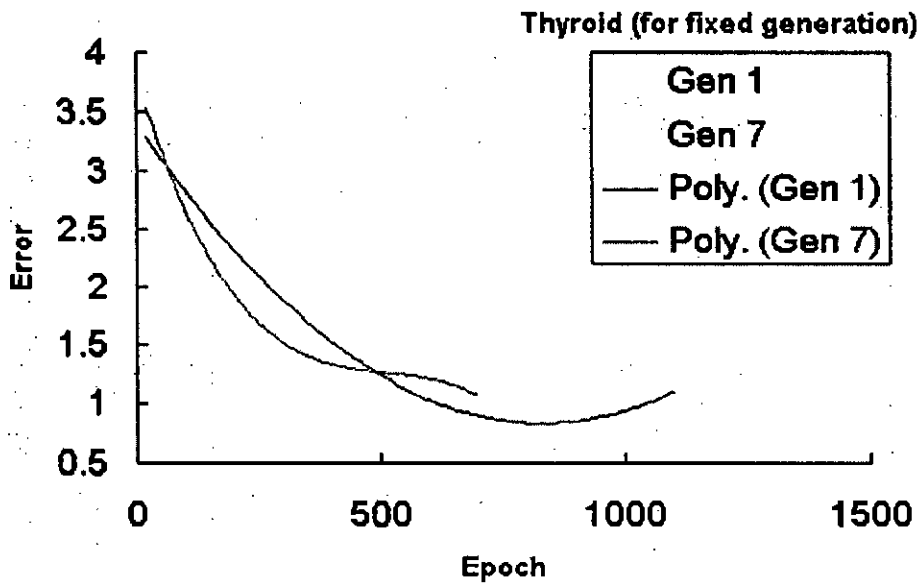


Figure 5.7: Epoch Vs Error for Thyroid.

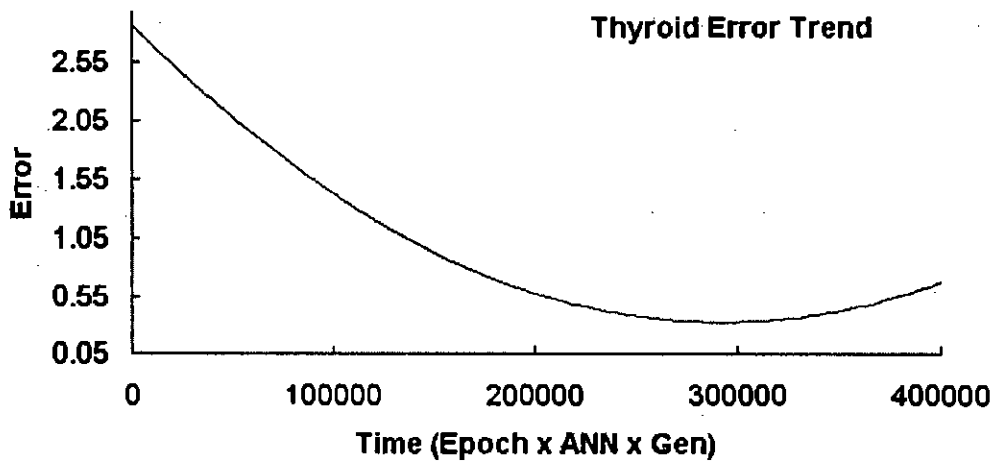


Figure 5.8: Time Vs Error for Thyroid.

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.

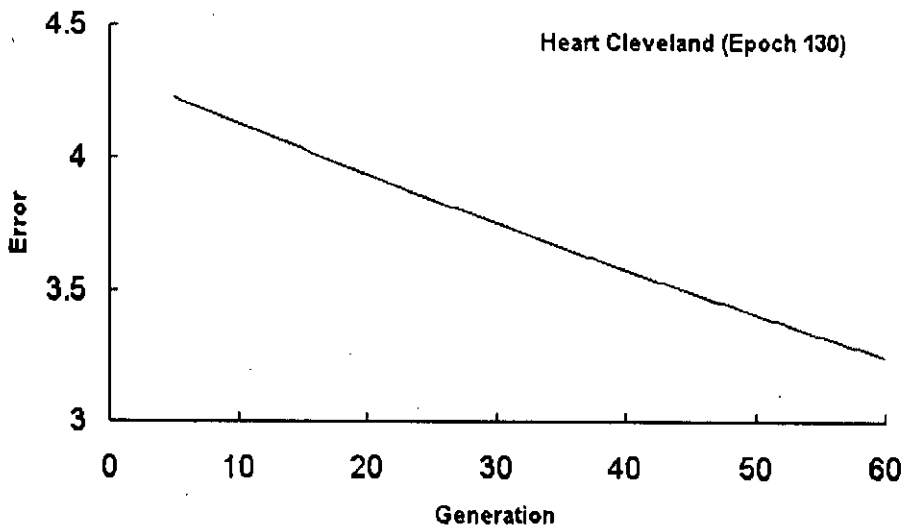


Figure 5.9: Generation Vs Error for heart disease.

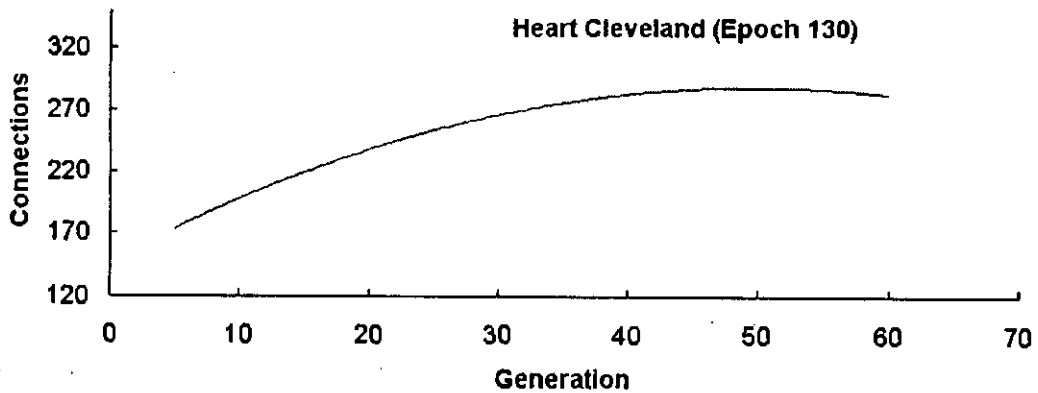


Figure 5.10: Generation Vs Connections for heart disease.

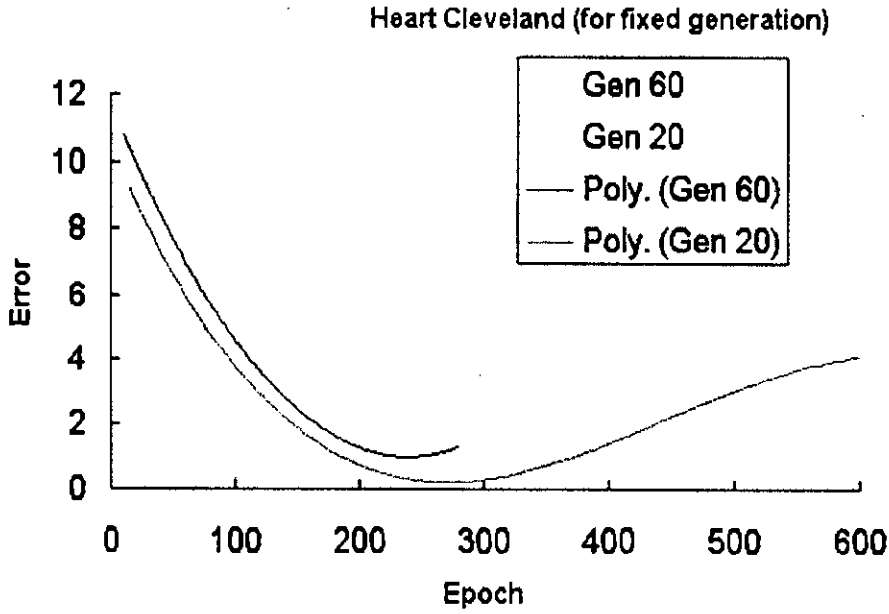


Figure 5.11: Epoch Vs Error for heart disease.

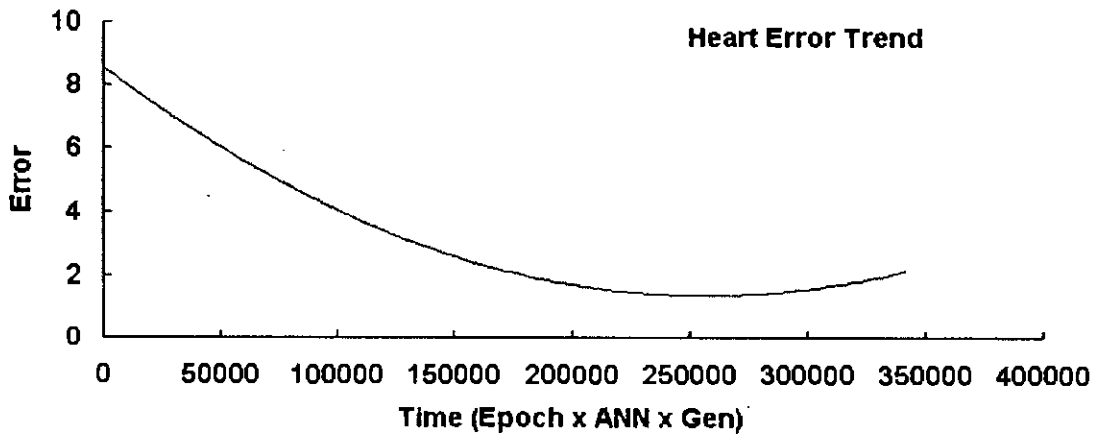


Figure 5.12: Time Vs Error for heart disease.

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.

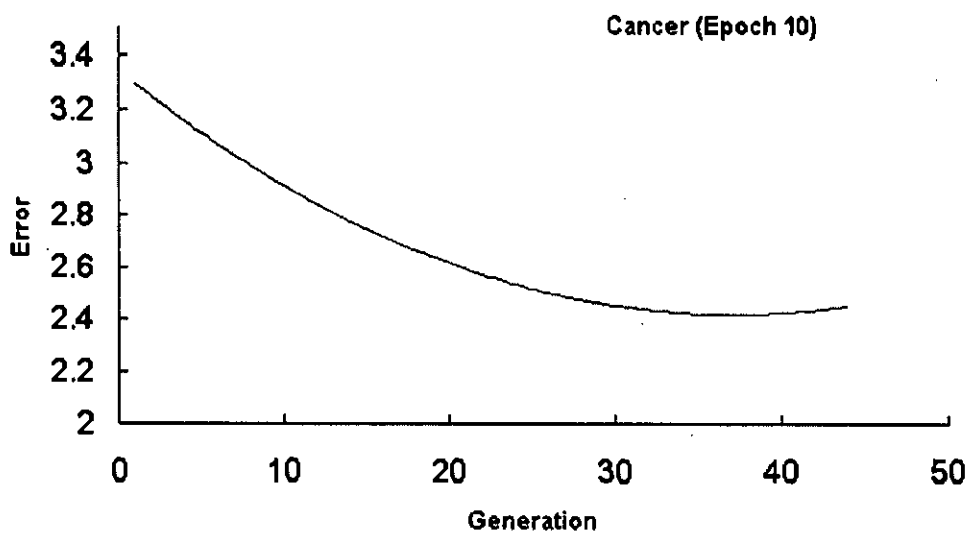


Figure 5.13: Generation Vs Error for breast cancer.

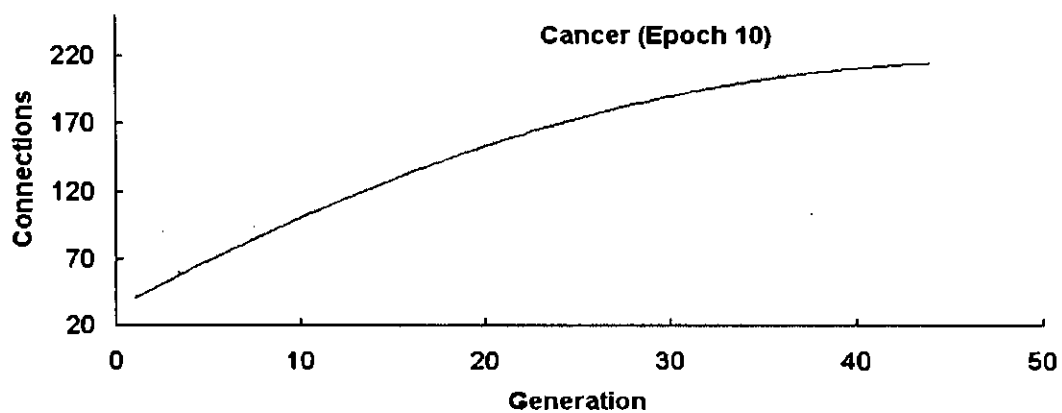


Figure 5.14: Generation Vs Connections for breast cancer.

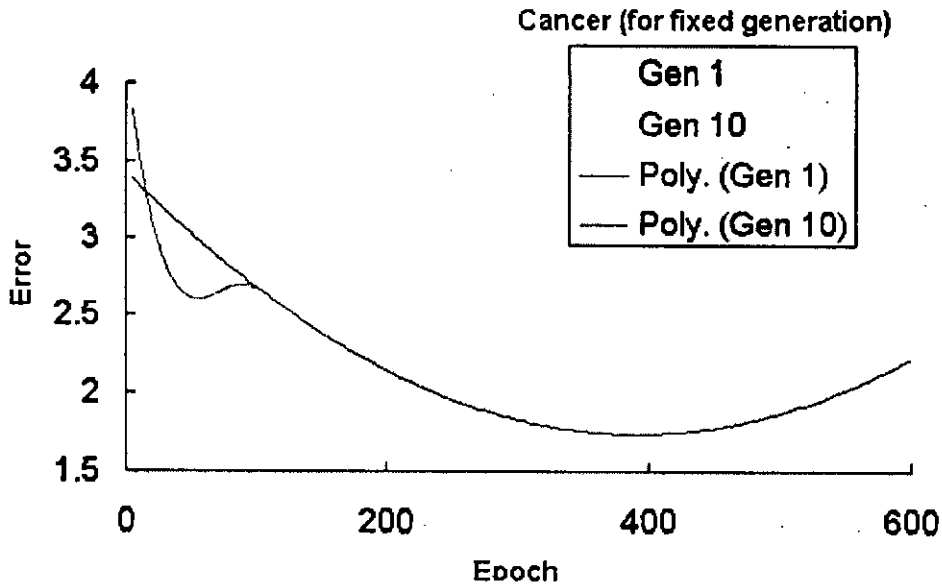


Figure 5.15: Epoch Vs Error for breast cancer.

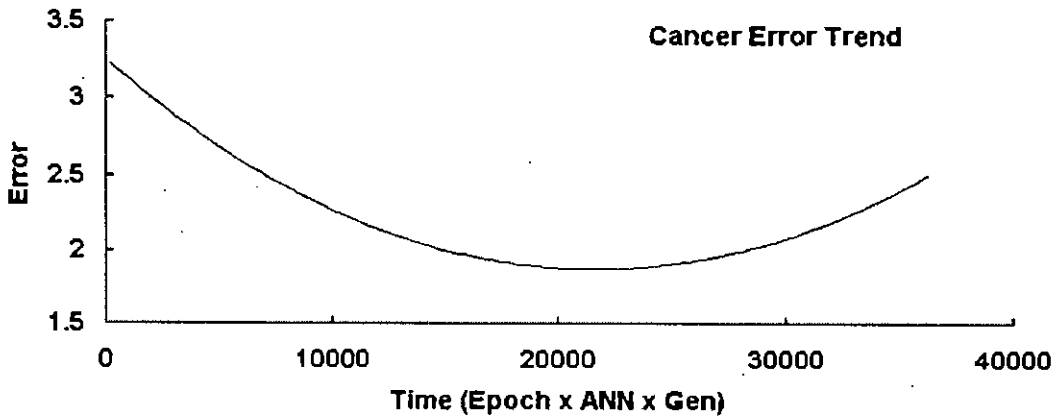


Figure 5.16: Time Vs Error for breast cancer.

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.

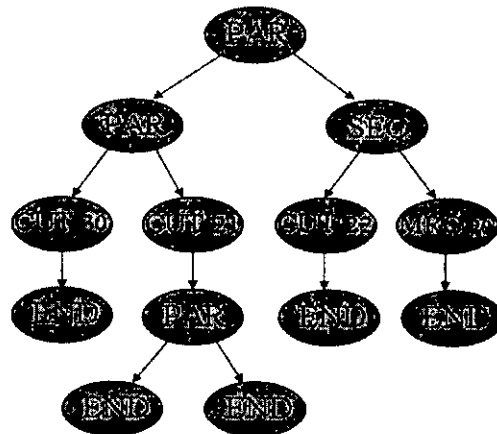


Figure 5.17: PST of the best ANN found in breast cancer problem.

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 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 use 27 disputable patterns, but here it 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
<i>EPNet</i>	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
<i>Our approach</i>	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
<i>EPNet</i>	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
<i>Our approach</i>	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
<i>EPNet</i>	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
<i>Our approach</i>	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
<i>EPNet</i>	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
<i>Our approach</i>	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	<i>RBF of GM algorithm</i>	<i>Bennet – Mangasarian (MSMI)</i>	<i>Bennet – Mangasarian (MSM)</i>	<i>Bennet – Mangasarian (BP)</i>	<i>Manually Designed ANN</i>	<i>Our approach</i>
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	<i>HDANNS by trial and error</i>	<i>Our approach</i>
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	<i>Prechelt</i>	<i>Our approach</i>
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	<i>Schiffmann et al. 21-20-3 net</i>	<i>Schiffmann et al. Multilayer ANN</i>	<i>Our approach</i>
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 adopted 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 Phenomena 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 Alope 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 Randolph 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](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.

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 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.

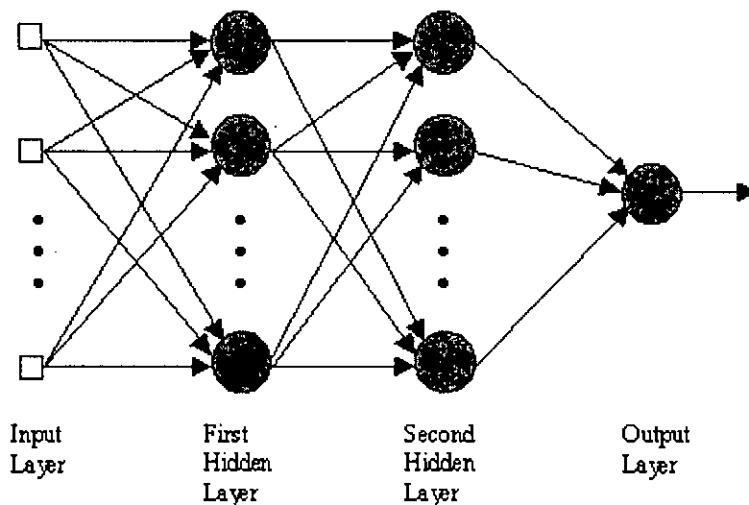


Figure A.1: A multilayer perceptron.

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 asks 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 cases 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.

