Intelligent Techniques for the Diagnosis of Coronary Artery Disease

Ravi Jain

This thesis submitted for the degree of

Doctor of Philosophy

Department of Applied Mathematics,
The University of Adelaide
South Australia

(November 1998)

Contents

List of Tables

List of Figures

Abstract

In this thesis, a genetic-programming-based classifier system for the diagnosis of coronary artery disease is proposed. It maintains good classification and generalisation performance. Based on genetic programming, a software system called Evolutionary Pre-Processor has been developed which is a new method for the automatic extraction of non-linear features for supervised classification. The central engine of Evolutionary Pre-Processor is the genetic program; each individual in the population represents a pre-processor network and a standard classification algorithm. The EPP maintains a population of individuals, each of which consists of an array of features. The features are transformations made up of functions selected by the user. A fitness value is assigned to each individual, which quantifies its ability to classify the data. This fitness value is based on the ability of a simple classifier to correctly classify the data after it has been transformed to the individual's feature space. Through the repeated application of recombination and mutation operators to the fitter members of the population, the ability of the individuals to classify the data gradually improves until a satisfactory point in the optimisation process is reached, and a solution is obtained.

Recently there has been a rising interest in using artificial intelligent (AI) techniques in the field of medical diagnosis. However, it is noted, that most intelligent techniques have limitations, and are not universally applicable to all medical diagnosis tasks. Each intelligent technique has particular computational properties, making them suitable for certain tasks over others. Integration of domain knowledge into empirical learning is important in building a useful intelligent system in practical domains since the existing knowledge is not always perfect and the training data are not always adequate. However, genetic algorithms (GAs) are robust but not the most successful optimization algorithms for any particular domain. Hybridizing a GA with algorithms currently in use can produce an algorithm better than both the GA and the current algorithms. A GA may be crossed with various problem specific search techniques to form a hybrid that exploits the global perspective of the GA and the convergence of the problem specific technique. In some cases, hybridization entails employing the representation as well as the optimization techniques already in use in the domain while tailoring the GA operators to the new representation.

In this connection a hybrid intelligent system is highly desirable. Here two different hybrid techniques are also presented. In the first approach, fuzzy systems is integrated with genetic algorithms. In this approach, each fuzzy if–then rule is treated as an individual and each population consists of certain number of fuzzy if then rules. It can automatically generate fuzzy if-then rule from training patterns for multi-dimensional for pattern classification problems. Classifiers in this approach are fuzzy if then rules. In the second approach genetic algorithms are combined with back-propagation algorithms to enhance the classification performance. In this approach, a complete set of weights and biases in a neural network are encoded in a string, which has an associated fitness indicating its effectiveness. Each chromosome completely describes a neural network. To evaluate the fitness of a chromosome, the weights on the chromosome are assigned to the links in a network of a given archtecture, the network is then run over the training set of examples and the sum of the sqaures of the errors is returned from each example.

All approaches were tested on a real-world problem of coronary artery disease data.

Statement of Originality

I hereby declare that this work contains no material which has been accepted for the award of any other degree or diploma in any University or other tertiary institution and that, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I also give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying.

Signed:

Date: : 30.11.18

Acknowledgments

This work wouldn't have been possible without the support of various people. At first, I wish to express my deep appreciation to my supervisor, Professor J. Mazumdar, for the invaluable encouragement and continued guidance and support received throughout the period under his supervision.

I am most grateful to Dr. P. Gill, Head, Department of Applied Mathematics, Dr. W. Venables, Head, Department of Statistics for their support and encouragement. My sincere thanks go to many reviewers who provided invaluable evaluations and recommendations (freely of their time) to read through the thesis, in part or in full.

Thanks also go to Dr. Jamie Sarrah, and Brandon Pincombe for reading the manuscript, Ian ball for their helpful comments on the written work. I would also to thank all my colleagues in the Department of Applied Mathematics at the University of Adelaide for their friendship and cooperation.

Finally, I would like to thank my parents and family for their moral support.

This thesis was supported financially with a University of Adelaide Scholarship, for which I am very grateful.

List of Publications

Proceedings

[1] Jain, L.C. and Jain, R.K.(Editors), Proceedings of the Second International Conference on Knowledge-Based Intelligent Engineering Systems, Volume 3, IEEE Press, USA, 1998.

[2] Jain, L.C. and Jain, R.K.(Editors), Proceedings of the Second International Conference on Knowledge-Based Intelligent Engineering Systems, Volume 2, IEEE Press, USA, 1998.

[3] Jain, L.C. and Jain, R.K.(Editors), Proceedings of the Second International Conference on Knowledge-Based Intelligent Engineering Systems, Volume 1, IEEE Press, USA, 1998.

Book

[1] Jain, L.C. and Jain, R.K. (Editors), " Hybrid Intelligent Engineering Systems " World Scientific Publishing Company, Singapore, March 1997.

Book Chapter

[1] Babri, H., Chen, L., Saratchandran, P., Mital D.P., Jain R. K., Johnson, R. P., Neural Networks paradigms, Soft computing techniques in Knowledge-based intelligent engineering systems, Springer Verlag, Germany, Chapter 2, pp 15-43, 1997.

Refereed Papers

[1] Jain, R.K., Mazumdar J., "Neural Network Approach for the Classification of Heart Disease Data" Australian Journal of Intelligent Information Processing Systems (to appear).

[2] Jain, R.K., Mazumdar J., Moran W. "Application of Fuzzy Classifier System to Coronary Artery Disease and Breast Cancer", Australasian Physical & Engineering Sciences in Medicine, vol. 21, Number 3, pp. 242-247, 1998.

[3] Sherrah J., Jain, R.K., "Classification of Heart Disease Data using the Evolutionary Pre-Processor" Engineering Mathematics and Application Conference, Adelaide, pp. 463-466, 1998.

[4] Jain, R.K., Mazumdar, J., and Moran, W.,"Expert System, Fuzzy Logic and Neural Networks in the Detection of Coronary Artery Disease", Journal of Biomedical Fuzzy and Human Sciences vol.2, pp. 45-54, 1996.

[5] Jain, R.K., Mazumdar, J., and Moran, W.,"Use of Multilayer perceptron in the classification of Coronary Artery Disease Data", Fourth International Conference on ICARCV'96, Westin Stamford Singapore, vol 3, pp. 2247-2251, 1996.

[6] Alpaslan, F.N. and Jain, R. K., Research in Biomedical sciences, ETD2000, IEEE Computer Society Press, Los Alamitos, USA, pp. 612-619, May 1995.

[7] Jain, R.K., Mazumdar, J., and Moran, W.,"A Comparative Study of Artificial Intelligent Techniques in the Detection of Coronary Artery Disease," ETD2000, IEEE Computer Society Press, Los Alamitos, USA pp. 113-120, 1995.

[8] Jain, R.K., "Neural Networks in Medicine," Artificial Neural Networks and Expert Systems, edited by N.K. Kasabov, IEEE Computer Society Press, Los Alamitos, USA, pp. 369-372, 1993.

[9] Kulshrestha, D.K., Pourbeik, P. and Jain, R.K., "Fault Diagnosis of Electronic Circuits using Neural Networks," Proceedings of Electronics '92, Adelaide, pp. 254-261, 1992.

Glossary of Terminology

CAD	Coronary Artery Disease
EPP	Evolutionary Pre-Processor
GA	Genetic Algorithms
GLIM	Generalised Linear Machine
GP	Genetic Programming
kNN	k-Nearest Neighbours
MDTM	Minimum-Distance-to-Means
ML	Gaussian Maximum likelihood
MLP	Multi-layer Perceptron
MN	Modular network
PPD	Parallelepiped
RBF	Radial Basis function
RL	Reinforcement Learning

Chapter 1

Introduction and Overview

1.1 Background to the Research

Coronary artery disease (CAD) is the most common cause of death in humans. It is a degenerative disease which is the result of an increase of atheroma in the coronary artery walls leading to total or partial occlusion. The resulting clinical feature is Myocardial Infarction and subsequently sudden death. The main risk factors related to this disease involve age, sex, chest pain type, resting blood pressure, cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise induce angina, ST depression induced by exercise relative to rest, slope of the peak exercise ST segment, smoking, hypertension, stress etc. This complex multifactorial disease makes it difficult for clinicians to accurately assess the likelihood of a cardiac event.

For this reason early detection of coronary artery disease is an important medical research area. One of the most reliable ways to diagnose coronary artery disease is cardiac catherization, leading to a final diagnosis (Watanabe, 1995). However, it differs from other method in that it is invasive, it requires a catheter to be inserted into a vein or artery and manipulated to the heart under radiographic fluoroscopic guidance. It is performed by a puncture or cut down of the brachial or femoral artery, from which the heart is approached retrogradely. Since some patients would die of an allergic shock to the angiographic enhancing agent used, they must be examined beforehand for such hypersensitivity by an intravenous injection of that agent. There is also the risk of life-threatening arrythmia, or irreversible invasion into the coronary arteries or other structures by the catheterization. If a sufficient high diagnostic rate can be assured at some stage prior to catheterization, it will be a very useful addition to the present medical diagnosis capability. Although accurate, it is costly and time consuming and there is always an element of risk.

The diagnosis of coronary artery disease is a complex decision making process. The electrocardiogram (ECG) is the principal diagnostic tool available at present but it often fails to diagnose the coronary artery disease. There is however a standard test that does give the correct diagnosis of coronary artery disease but this involves the measurement of enzyme and ECG changes over a period of 24 to 48 hours. This method offers no help in the early diagnosis of coronary artery disease.

A considerable number of methodologies have been developed to analyze clinical data collected during patient evaluation in attempts to improve on the diagnostic accuracy of physicians in identifying coronary artery disease. But none of these approaches has been able to improve significantly on clinical data. Conventional

statistical multivariate analysis methods are used to measure risk on those asymtomatic patients who, nevertheless, present one, two or more CAD risk factors. Although statistical methods provide information regarding the likelihood of CAD event through systematic numerical calculation, it does not take in account the individual, neither does it provide sophisticated human-like reasoning.

It is obvious that there is a need by which the data available in the clinical setting can be analyzed to yield information that can be utilized to assist the clinician in making a fast and accurate diagnosis of coronary artery disease. An intelligent approach using artificial neural networks, genetic algorithms and fuzzy logic can assist the clinician for this purpose.

The main objective of this research is to develop the intelligent techniques in diagnosing coronary artery disease from a given set of patients data. The techniques developed here include genetic programming in order to evolve optimal subsets of discriminatory features for robust pattern classification and hybrid learning methodology that integrates artificial neural networks, genetic algorithms, fuzzy systems.

1.2 An introduction to Coronary Artery Disease

The term coronary artery disease (CAD) (Figure 1-1) refers to degenerative changes in the coronary circulation. Cardiac muscles fibres need a constant supply of oxygen and nutrients and any reduction in the coronary circulation produces a corresponding reduction in the cardiac performance. Such reduced circulatory supply known as coronary ishemia (is-KE-me-a) usually results from partial or complete blockage of the coronary arteries. Figure 1-2 shows a myocardial perfusion scan and coronary arteriogram in the presence of a mild stenosis of the left anterior descending coronary artery.

The usual cause is the formation of fatty deposit, or plaque, in the wall of a coronary vessel. The plaque, or an associated thrombus, then narrows the passageway and reduces blood flow. In myocardial infarction (MI) or heart attack the coronary circulation becomes blocked and the cardiac muscle cells die from lack of oxygen. The affected tissue then degenerates, creating a non-functional area known as an infarct. Heart attacks most often result from severe coronary artery disease. The consequences depend on the site and nature of the circulatory blockages.

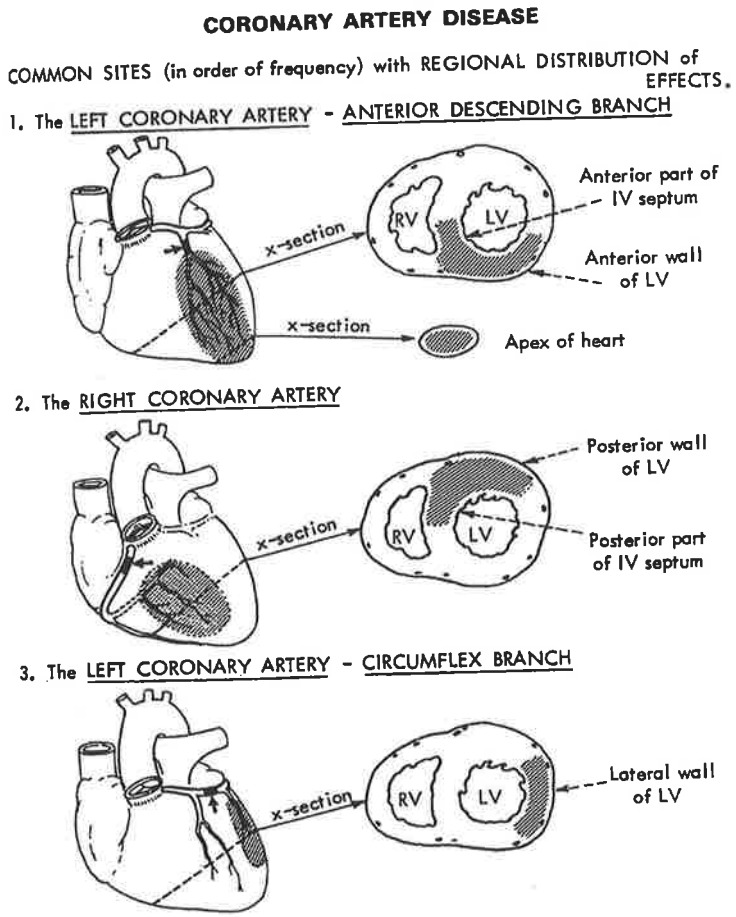


Figure 1-1: Coronary Artery Disease (adapted from Pathology Illustrated)

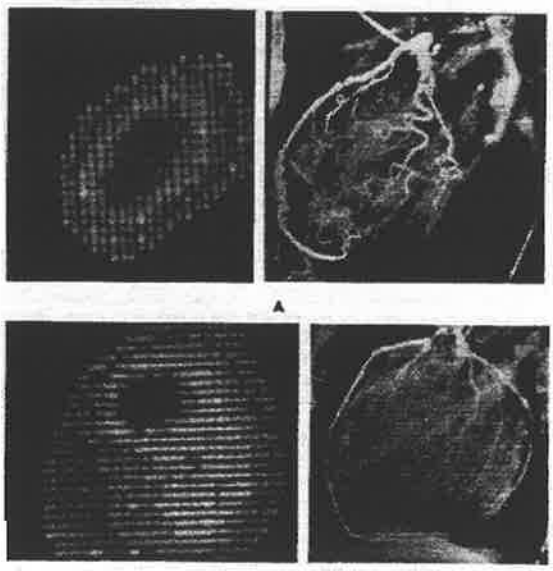


Figure 1-2: A Myocardial perfusion scan and coronary arteriogram in the presence of a mild stenosis of the left anterior descending coronary artery (mid portion).

1.3 Rationale of this Thesis

This thesis has the following objectives:

1. A genetic-programming-based classifier system for the diagnosis of coronary artery disease (Chapter 8).
2. A comparison of genetic programming with various well-known classification techniques.
3. A concise overview of pattern recognition (Chapter 2).
4. A concise overview of Neural Networks and genetic algorithms (Chapter 3, Chapter 4).
5. Use of multilayer perceptron for the diagnosis of coronary artery disease (Chapter 6).
6. Use of neuro/genetic algorithm for the diagnosis of coronary artery disease (Chapter 7).
7. Use of genetic/fuzzy classifier system for the diagnosis of coronary artery disease (Chapter 9).

Furthermore, in the thesis the effects of the input clinical variables for the diagnosis of coronary artery disease using neural network, a hybrid of genetic and the back-propagation algorithms, and fuzzy and genetic algorithms are evaluated. It is shown that these AI methods give best diagnostic performance than the other methods. It is shown in this study that neural network appears to place diagnostic importance on certain clinical variables.

It is obvious from brief literature review that there is not a single technique which has answer to all the problems. Thus it is useful to integrate various artificial intelligence based techniques such as, neural networks and genetic algorithms (GAs) and fuzzy systems and GAs.

1.4 Artificial Neural Networks, Genetic Algorithms, Fuzzy System

Artificial neural networks and genetic algorithms are the examples of microscopic biological models. They originated from modelling of the brain and evolution. ANNs are a new generation of information processing systems. The main theme of neural network research focuses on modelling of the brain as a parallel computational device for various computational tasks that were performed poorly by traditional serial computers. They have a large number of highly interconnected processing nodes that usually operate in parallel. ANNs, like a human brain, demonstrates the ability to learn, recall, and generalise from training patterns. The ANNs offer a number of advantages over the conventional computing techniques such as the ability to learn arbitrary non-linear input-output mapping directly from training data. They can sensibly interpolate input patterns that are new to the network. Neural networks can automatically adjust their connection weights or even network structures. The inherent fault-tolerance capability of neural network stems from the fact that the large number of connections provides much redundancy, each node acts independently of all the others, and each node relies only on local information.

Genetic algorithms (GAs) were developed by John Holland in the 1970s. These algorithms are general-purpose search algorithms based on the principles of natural population genetics. The GA maintains a population of strings, each of which

represents a solution to a given problem. Every member of the population is associated with a certain fitness value, which represents the degree of correctness of that particular solution. The initial population of strings is randomly chosen, and these strings are

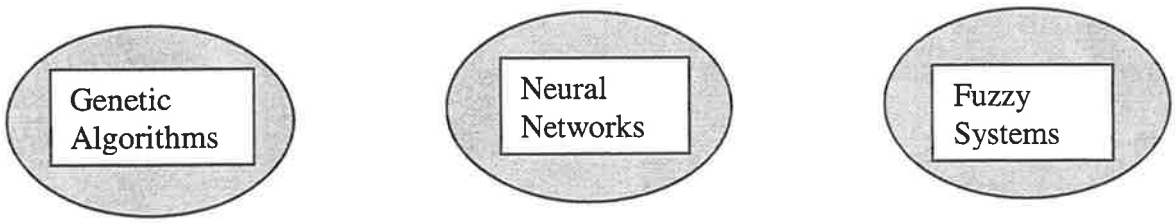


Figure 1-3: Soft Computing

manipulated by the GA using genetic operators to finally arrive at the best solution to the given problem. The main advantage of a GA is that it is able to manipulate a large population of strings at the same time, each of which represents a different solution to the given problem. This way the possibility of the GA getting stuck in local minima is greatly reduced, because the whole space of possible solutions can be searched simultaneously. A basic genetic algorithm comprises three genetic operators:

- selection,
- crossover, and
- mutation

Starting from the initial population of strings the GA uses these operators to calculate successive generations. First, pairs of individuals of the current population are selected to mate with each other to form the offspring, which then form the next generation.

Selection is based on the survival of the fittest strategy, but the key idea is to select the better individuals of the population as in tournament selection, where the participants compete with each other to stay in the competition. The most commonly used strategy to select pairs of individuals is the method of roulette-wheel selection, in which every string is assigned a slot in the wheel sized in proportion to the string's relative fitness. This ensures that highly fit strings have a greater probability to be selected to form the next generation through crossover and mutation. After selection of the pairs of parent strings the crossover operator is applied to each of these pairs.
The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. In single point crossover a bit position along the two strings is selected at random and the two parent strings exchange their genetic material as illustrated below.

Parent A = $a_1\ a_2\ a_3\ a_4 \mid a_5\ a_6$
Parent B = $b_1\ b_2\ b_3\ b_4 \mid b_5\ b_6$

The swapping of genetic material between the two parents on either side of the selected crossover point produces the following offspring:

Offspring $A' = a_1\ a_2\ a_3\ a_4 \mid b_5\ b_6$

Offspring $B' = b_1\ b_2\ b_3\ b_4 \mid a_5\ a_6$

The two individuals (children) resulting from each crossover operation will now be subjected to the mutation operator in the final step to forming the new generation.

The mutation operator alters one or more bit values at randomly selected locations in randomly selected strings. Mutation takes place with a certain probability, which in accordance with it's biological equivalent is usually a very low probability. The mutation operator enhances the ability of the GA to find a near optimal solution to a given problem by maintaining a sufficient level of genetic variety in the population, which is needed to make sure that the entire solution space is used in the search for the best solution.

1.5 Fuzzy Systems

The fuzzy logic was developed by Lotfi Zadeh (1965) to provide approximate but an effective means of describing the behavior of the systems that are not easy to tackle mathematically. Zadeh stated that as the complexity of the system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and relevance become almost mutually exclusive characteristics. Attempts to automate various types of activities including diagnosing a patient have been impeded by the gap between the way human reasons and the way computers are programmed. Thus the fuzzy logic is a step towards automation. Inspite of many advantages, fuzzy logic has few problems. These are, for example, the lack of design procedure for determining membership function and the lack of design adaptability for possible changes in the reasoning environment. The use of artificial neural networks in designing the membership function reduces system development time and the cost and increases the performance of the system. Traditionally, the users set the structure of a feed-forward neural network a priori. The type of the structure used may perhaps be based on some knowledge of the medical diagnostic problem but usually the neural network structure is found by the trial and error method. In many cases the structure is a fully connected feed forward neural network and the users might try to vary the number of neurons in the hidden layers. Such a network structure is then trained using a suitable learning algorithm to generate an optimal set of weights while the structure is taken for granted or chosen from limited domain. The method of trial and error is not only time consuming but may not generate an optimal structure. It is possible to use genetic algorithm in the automatic generation of neural network structure and optimise its weights.

1.6 Outline of this Thesis

The thesis contains the following chapters.

The work begins in Chapter 2 with an introduction to pattern recognition. The classification methods used for the experimental work are described. Difference between Perceptron and Gaussian classifiers also presented.

Chapter 3 presents a detail description of single layer perceptron, and multilayer perceptron (MLP). In this chapter limitations of perceptron, learning techniques and mathematics of MLP, are highlighted.

Chapter 4 is an introduction to the genetic algorithms (GAs). In this chapter mathematical foundation of genetic algorithm, genetic operations etc. are described and compared it with the gradient descent algorithm.

Chapter 5 describes the fuzzy set theory. It presents principal concepts and mathematical notions of fuzzy set theory, fuzzy set operations, properties of fuzzy sets, etc. The purpose of this chapter is to build the sound theoretical background which is necessary to design fuzzy systems. This chapter forms a basis for Chapter 9.

Chapter 6 presents experiments and results of MLP. This chapter begins with a description of the data sets used to test the algorithm. In this chapter an attempt has been made to formulate the neural network training criteria in medical diagnosis. Also, the results are compared with various neural networks such as modular networks, radial basis function, reinforcement learning.

Chapter 7 describes the experimental results of hybrid system. In this chapter, the training of MLP by the GA optimization method is presented. It involves the optimization of connection weights of MLP architecture for solving a specified mapping of an input data set to output data set. Also, it is compared with the back-propagation algorithm. Experimental results are presented using 1-point crossover, 2-point crossover, uniform crossover, NN-specific 2-point crossover, and NN-specific uniform crossover.

Chapter 8 presents the experimental results of genetic programming. Five simple statistical classification techniques are used, and the results are compared.

Chapter 9 presents the experimental results of genetic-algorithm-based fuzzy classifier system. Here each fuzzy if-then rule was treated as an individual. The fitness value of each fuzzy if-then was determined by the numbers of correctly and wrongly classified training patterns by that rule.

In Chapter 10 the conclusions are presented about the overall research topic, followed by implications for the larger field of research. This chapter ends with suggestions for future research.

Finally, there are three appendices. Appendix A contains the generalised delta rule for Multilayer Perceptron. Appendix B contains neural network learning using GAs. Appendix C contains codes for the design of fuzzy/genetic classifiers respectively.

1.7 Conclusion

This chapter has presented an overview of this thesis. It has established that CAD is a major problem. The present methods of diagnosis have been described and problems with them identified. It also presents brief introduction to artificial neural networks, fuzzy systems and hybrid intelligent techniques. These methods outperform the present methods of diagnosis in several important ways. The background of the research problem was introduced and the basic methodology used for the research was described.

Chapter 2

Pattern Recognition

2.1 What is pattern recognition?

Pattern recognition can be defined as a process of identifying structure in data by comparisons to known structure; the known structure is developed through methods of classification. For example, a child learns to distinguish the visual patterns of mother and father, the aural patterns of speech and music. A mathematician detects patterns in mathematics. By finding structure, we can classify the data according to similar patterns, attributes, features, and other characteristics. Any object or pattern that has to be classified must posses a number of discriminatory features. The first step in any recognition process, performed either by a machine or by a human being, is to choose candidate discriminatory features and evaluate them for their usefulness. Feature selection in pattern recognition involves the derivation of salient features from the raw input data in order to reduce the amount of data used for classification and simultaneously provide enhanced discriminatory power. The number of features needed to successfully perform a given classification task depends on the discriminatory qualities of the selected features. The input to a pattern recognition machine is a set of N measurements and the output is the classification. We represent the input by N dimensional vector x, called a pattern vector, with its components being N measurements. The classification at the output depends on the input vector x. Different input observations should be assigned to the same class if they have similar features and to different features if they have dissimilar features. The data used to design a pattern recognition system are usually divided into two categories: training data and test data. Discriminant functions are the basis for the majority of pattern recognition problems.
Pattern classification techniques fall into two categories.

1. Parametric
2. Non-parametric

A parametric approach to pattern classification defines discriminant function by a class of probabilities densities defined by a relatively small number of parameters. In fact, all parametric methods in both pattern classification and feature selection assume that each pattern class arises from a multivariate Gaussian distribution, where the parameters are the mean and covariances.

Numeric techniques include deterministic and statistical measures. These can be considered as measures made on geometric pattern space. In the statistical approach to numerical pattern recognition each input observation is represented as a multi-dimensional data vector. Statistical pattern recognition systems rest on mathematical models.

Non-numeric techniques are those that include the domain of symbolic processing that is dealt with by such methods as fuzzy sets.

2.2 Pattern Classification

Any object/pattern that has to be recognised or classified must possess a number of discriminatory properties or features. The first step in any recognition process, performed either by a machine or by a human being, is to choose candidate discriminatory properties or features and evaluate them for their usefulness. Feature selection in pattern recognition involves the derivation of silent features from the raw input data in order to reduce the amount of data used for classification and simultaneously provide enhanced discriminatory power. The number of features needed to successfully perform a given classification task depends on the discriminatory qualities of the selected features.

The traditional form of a pattern classifier is shown in Figure 2-1. The original data measurements are fed into the pre-processor $\mathbf{x}_i$ which outputs features $\mathbf{y}_i$. These features are then fed to the classifier, which outputs a class label c_i'. For the training samples with known class labels c_i, the difference between c_i' and c_i and are used to train the classifier. For novel samples, c_i' is the predictor for the class of that sample.

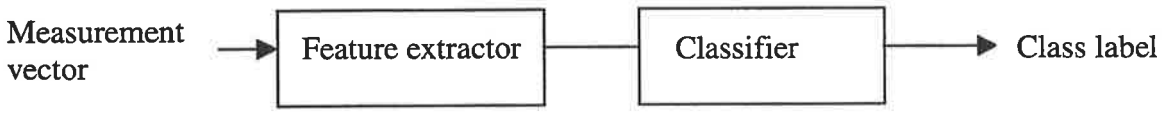


Figure 2-1: A pattern recognition system

2.3 Minimum error rate Classification Error

In classification problems, each state of nature is usually associated with a different one of the c classes and the action α_i is usually interpreted as the decision that the true state of nature is ω_i. If action α_i is taken and the true state of nature is ω_j then the decision is correct if i=j, and an error if i≠j. The job of the classifier is to find an optimal decision that will minimize the average probability of error i.e. risk or error rate. A loss function so called *symmetrical or zero-one* loss function assigns no loss to a correct decision, and assigns a unit loss to any error. Thus all errors are equally costly. The risk corresponding to this loss function is precisely the average probability of error, since the conditional risk is

$$\begin{aligned} R(\alpha_i|\mathbf{x}) &= \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) P(\omega_j|\mathbf{x}) \\ &= \sum_{j\neq i} P(\omega_j|\mathbf{x}) \\ &= 1 - P(\omega_i|\mathbf{x}) \end{aligned} \tag{2.1}$$

where

$$\lambda(\alpha_i|\omega_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \qquad i, j = 1,\ldots,c. \tag{2.2}$$

and $P(\omega_i|\mathbf{x})$ is the conditional probability that action α_i is correct. The Bayes decision rule to minimize risk calls for selecting the action that minimizes the conditional risk.

Thus to minimize the average probability of error we should select the i that *maximizes* the a posteriori probability $P(\omega_i|\mathbf{x})$. In other words, *for minimum error rate*:

$$\text{Decide } \omega_i \quad P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}) \; \forall \; j \neq i.$$

2.4 Bayesian Classification

The Bayesian approach is an analytical method and is very powerful and widely used. Under this approach the problem is posed in probabilistic terms and all the probabilities and distributions are assumed to be known (Duda and Hart, 1973). The advantages of this approach are that it is theoretically well-founded empirically well-proven and involves procedural mechanisms whereby new problems can be systematically solved. It relies on the basic statistical theory of probabilities and conditional probabilities.

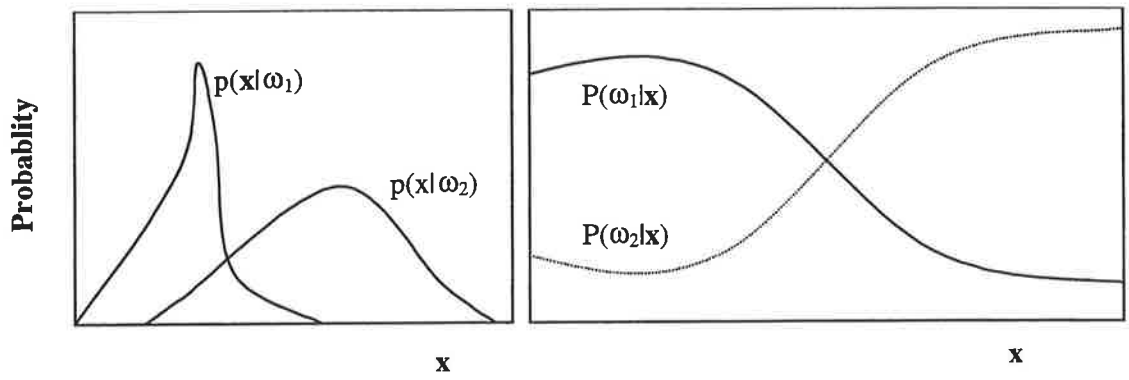


Figure 2-2: Conditional probability density functions and a posteriori probabilities.

The pivotal mathematical tool for this analysis is *Bayes Rule*:

$$P(\omega_j) = \frac{p(\mathbf{x} \mid \omega_j)P(\omega_j)}{p(\mathbf{x})} \tag{2.3}$$

where

$$p(\mathbf{x}) = \sum_{j=1}^{2} p(\mathbf{x} \mid \omega_j)P(\omega_j) \tag{2.4}$$

Bayes rule shows by observing the value of x changes the a priori probability $P(\omega_j)$ to the a *posteriori probability* $P(\omega_j|\mathbf{x})$. For example, if we have an observation x for which $P(\omega_1|\mathbf{x})$ is greater than $P(\omega_2|\mathbf{x})$, we would normally decide that the true state of nature is ω_1. Similarly, if $P(\omega_2|\mathbf{x})$ is greater than $P(\omega_1|\mathbf{x})$ we would decide to choose ω_2. For **x**, the probability of error is defined as follow.

$$P(\text{error} \mid x) = \begin{cases} P(\omega_1(\mathbf{x})) & \text{if we decide } \omega_2 \\ P(\omega_2(\mathbf{x})) & \text{if we decide } \omega_1 \end{cases} \tag{2.5}$$

We can minimize the probability of error by deciding ω_1 for the same value of x if $P(\omega_1|\mathbf{x}) > P(\omega_2|\mathbf{x})$ and ω_2 if $P(\omega_2|\mathbf{x}) > P(\omega_1|\mathbf{x})$. The average probability of error is given by

$$\begin{aligned} P(\text{error}) &= \int_{-\infty}^{\infty} P(\text{error}, \mathbf{x})\, dx \\ &= \int_{-\infty}^{\infty} P(\text{error}, \mathbf{x}) p(\mathbf{x})\, dx \end{aligned} \tag{2.6}$$

Figure 2-2 shows the difference between $p(x|\omega_1)$ and $p(x|\omega_2)$ and the variations of $P(\omega_j|x)$ with x. Let $P(\omega_j)$ be the *a priori probabilities* and $p(x|\omega_j)$ be the *state-conditional probability density* function for x, the probability density function for x given that the state of nature is ω_j.

To summarise, the overall Bayesian approach for minimising error rate is to estimate or hypothesise the priors and the conditional class densities, then invert these to obtain the posteriors. For a new x, the posterior with maximum value corresponds to the Bayes optimal class. The freedom in choice of learning algorithms is in the way the conditional probability density functions are formed. Some common methods are discussed here.

2.5 Linear Discriminant Functions

A linear discriminant function divides the feature space by a hyperplane decision surface. The orientation of the surface is determined by the normal vector **w**, and the location of the surface is determined by the threshold weight w_0. The discriminant function $g(\mathbf{x})$ is proportional to the signed distance from **x** to the hyperplane, with $g(\mathbf{x}) > 0$ when **x** is on the positive side, and $g(\mathbf{x}) < 0$ when **x** is on the negative side.

A discriminant function that is linear combination of the components of **x** can be written as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0, \tag{2.7}$$

where w is called the *weight vector* and w_0 *the threshold weight*. A two category linear classifier implements the following decision rule: Decide ω_1 if $g(\mathbf{x}) > 0$ and ω_2 if $g(\mathbf{x}) < 0$.

Thus, x is assigned to ω_1 if the inner product $\mathbf{w}^t\mathbf{x}$ exceeds the threshold $-w_0$. If $g(\mathbf{x}) = 0$, **x** can be assigned to either class.

The equation $g(\mathbf{x}) = 0$ defines the decision surface that separates points assigned to ω_1 from points assigned to ω_2. When $g(\mathbf{x})$ is linear, this decision surface is a *hyperplane*. If x_1 and x_2 are both on the decision surface, then

$$\begin{aligned} \mathbf{w}^t\mathbf{x}_1 + w_0 &= \mathbf{w}^t\mathbf{x}_2 + w_0 \\ \text{or} \qquad \mathbf{w}^t(\mathbf{x}_1 - \mathbf{x}_2) &= 0, \end{aligned} \tag{2.8}$$

so that w is normal to any vector lying in the hyperplane. In general the hyperplane divides the feature space into two halfspaces, the decision region R_1 for ω_1 and the decision region R_1. Since $g(\mathbf{x}) > 0$ if x is in R_1, it follows that the normal vector w points into R_1.

The discriminant function g(x) gives an algebraic measure of the distance from x to the hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}, \tag{2.9}$$

where xp is the normal projection of **x** onto H, and r is the desired algebraic distance, positive if x is on the positive side and negative if x is on the negative side. Then $g(\mathbf{x}_p) = 0$,

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x}_1 + w_0 = r\|\mathbf{w}\|, \tag{2.10}$$

An illustration of these results are given in Figure 2-3.

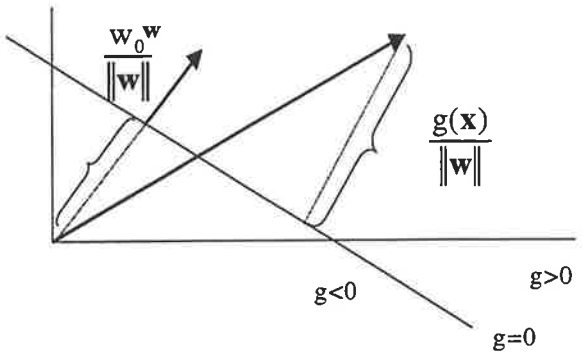


Figure 2-3: The linear decision boundary $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$

2.6 Quadric discriminant function

A quadric discriminant function has the form

$$g_i(\mathbf{X}) = \sum_{j=1}^{d} w_{jj}x_j^2 + \sum_{j=1}^{d-1}\sum_{k=j+1}^{d} w_{jk}x_j x_k + \sum_{j=1}^{d} w_j x_j + w_{d+1} \tag{2.11}$$

Any machine, which employs quadric discriminant function, is called a *quadric machine* (see Figure 2-4). A quadric discriminant function has (d+1)(d+2)/2 parameters or weights consisting of

d weights as coefficients of x_j^2 terms	w_{jj}
d weights as coefficients of x_j terms	w_j
d(d-1) weights coefficients of x_j x_k terms, $k \neq j$	w_{jk}
1 weight which is not a coefficient	w_{d+1}

Equation (2.11) can be put into matrix form after making the following definitions. Let the matrix $\mathbf{A} = [a_{jk}]$ have the following components given by

$a_{jj} = w_{jj}$ $\quad j = 1, \ldots, d$
$a_{jk} = \frac{1}{2}w_{jk}$ $\quad j, k = 1, \ldots, d,\ j \neq k$

Let the column vector $\mathbf{B} = \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_d \end{pmatrix}$ have components given by $b_j = w_j$, $j = 1, \ldots, d$.

Let the scalar $C = w_{d+1}$. Then

$$g(\mathbf{X}) = \mathbf{X}^t \mathbf{A} \mathbf{X} + \mathbf{X}^t \mathbf{B} + C \tag{2.12}$$

where $\mathbf{X}$ is a column vector and $\mathbf{X}^t$ denotes the transpose of X. The term $\mathbf{X}^t \mathbf{A} \mathbf{X}$ is called a *quadric form*. If all the eigenvalues of $\mathbf{A}$ are positive, the quadric form is never negative for any vector $\mathbf{X}$ and equal to zero only for

$$X = \begin{pmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix}$$

When these conditions are met both the matrix $\mathbf{A}$ and the quadric from are called *positive definite*. If $\mathbf{A}$ has one or more of its eigenvalues equal to zero and all the others positive, then the quadratic form will never be negative, and it and $\mathbf{A}$ are called *positive semidefinite.*

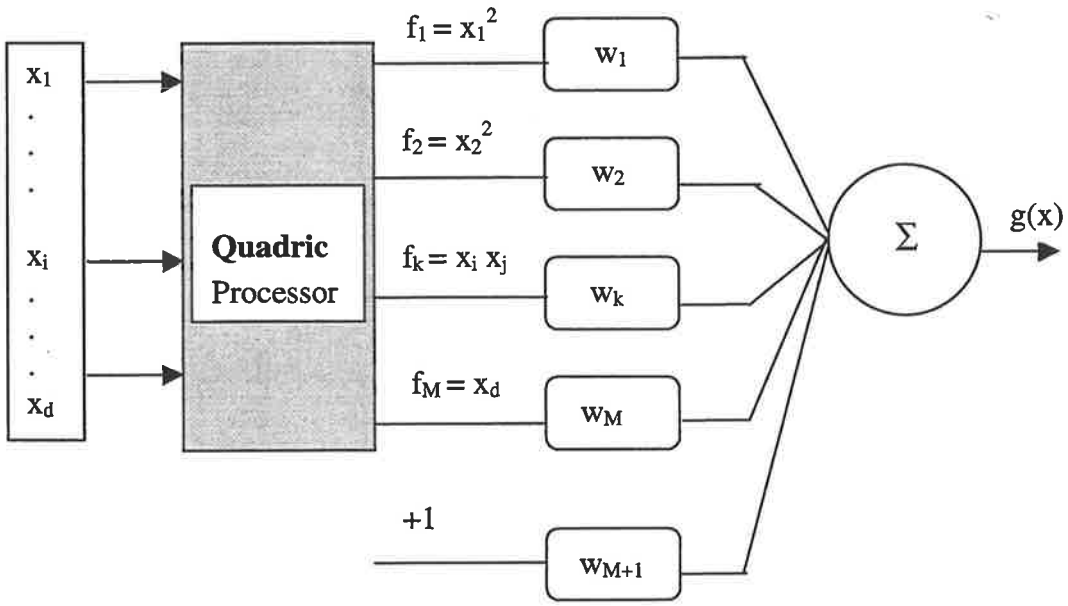


Figure 2-4: A quadric discriminant machine

2.7 Maximum-Likelihood Gaussian Classifier

A single-layer perceptron and the maximum-likelihood Gaussian classifier are both examples of linear classifiers. The decisions regions formed by perceptron are similar to those the maximum-likelihood Gaussian classifier which assume inputs are uncorrelated and distributions for different classes differ in mean values.
The maximum-likelihood Gaussian Classifier minimizes the average probability of classification error. This minimization is independent of the overlap between the underlying Gaussian distributions of the two classes.

The maximum-likelihood method is a classical parameter-estimation method that views the parameters as quantities whose values are fixed but unknown. The best estimate is defined to be the one that maximizes the probability of obtaining the samples. The observation vector $\mathbf{x}$ is described in terms of *mean vector* μ and *covariance matrix* $\mathbf{C}$ which are defined by, respectively.

$$\mu = E[\mathbf{x}] \tag{2.13}$$

and

$$\mathbf{C} = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T) \tag{2.14}$$

where E is the expected value. Assuming that the vector $\mathbf{x}$ is *Gaussian-distributed*, the *joint-probability density* function of the element of x as follows.

$$f(x) = \frac{1}{(2\pi)^{p/2}(\det \mathbf{C})^{1/2}} e^{\left[-\frac{1}{2}(x-\mu)^T \mathbf{C}^{-1}(x-\mu)\right]} \tag{2.15}$$

where det C is the determinant of the covariance matrix C, the matrix C^{-1} is the inverse of $\mathbf{C}$, and p is the dimension of the vector $\mathbf{x}$.

Suppose that the vector $\mathbf{x}$ has a mean vector the value of which depends on whether it belongs to class ω_1 or class ω_2, and a covariance matrix that is the same for both classes. Furthermore we may assume that the classes ω_1 and ω_2 have equal probability and the samples of both classes are correlated, so that the covariance matrix $\mathbf{C}$ is non-diagonal, and it is non-singular. Then the joint-probability density function of the input vector $\mathbf{x}$ can be expressed as follow:

$$f(x|\omega_i) = \frac{1}{(2\pi)^{p/2}(\det \mathbf{C})^{1/2}} e^{\left[-\frac{1}{2}(x-\mu)^T \mathbf{C}^{-1}(x-\mu)\right]} \tag{2.16}$$

Taking the natural logarithm of both sides of (2.16), and expanding terms, we get

$$\ln f(\mathbf{x}|\omega_i) = \frac{-p}{2}\ln(2\pi) - \frac{1}{2}(\det \mathbf{C}) - \frac{1}{2}\mathbf{x}^T\mathbf{C}^{-1}\mathbf{x} + \mu_i{}^T\mathbf{C}^{-1}\mathbf{x} - \frac{1}{2}\mu_i{}^T\mathbf{C}^{-1}\mu_i \tag{2.17}$$

$$l_i(\mathbf{x}) = \mu_i{}^T\mathbf{C}^{-1}\mathbf{x} - \frac{1}{2}\mu_i{}^T\mathbf{C}^{-1}\mu_i \tag{2.18}$$

For i=1, 2.

Hence a log-likelihood for class 1 can be defined as follows:

$$l_1(\mathbf{x}) = \mu_1{}^T \mathbf{C}^{-1}\mathbf{x} - \frac{1}{2}\mu_1{}^T \mathbf{C}^{-1}\mu_1 \tag{2.19}$$

Similarly a log-likelihood for class 2 can be defined as follows:

$$l_2(\mathbf{x}) = \mu_2{}^T \mathbf{C}^{-1}\mathbf{x} - \frac{1}{2}\mu_2{}^T \mathbf{C}^{-1}\mu_2 \tag{2.20}$$

Hence subtracting equation (2.20) from (2.19)

$$l = l_1(\mathbf{x}) - l_2(\mathbf{x}) \tag{2.21}$$

$$= (\mu_1 - \mu_2)^T \mathbf{C}^{-1}\mathbf{x} - \frac{1}{2}(\mu_1{}^T \mathbf{C}^{-1}\mu_1 - \mu_2{}^T \mathbf{C}^{-1}\mu_2) \tag{2.22}$$

which is linearly related to the input vector x. Rewriting the above equation

$$l = \hat{\mathbf{w}}^T \mathbf{x} - \hat{\theta} \tag{2.23}$$

$$= \sum_{i=1}^{p} \hat{\mathbf{w}}_i \mathbf{x}_i - \hat{\theta} \tag{2.24}$$

where $\hat{\mathbf{w}}$ is the *maximum-likelihood estimate*[1] of the known parameter vector $\hat{\mathbf{w}}$, defined by

$$\hat{\mathbf{w}} = \mathbf{C}^{-1}(\mu_1 - \mu_2) \tag{2.25}$$

and $\hat{\theta}$ is a constant threshold defined by

$$\hat{\theta} = \frac{1}{2}(\mu_1{}^T \mathbf{C}^{-1}\mu_1 - \mu_2{}^T \mathbf{C}^{-1}\mu_2) \tag{2.26}$$

Neither the perceptron nor the maximum-likelihood Gaussian classifier is appropriate when classes can not be separated by a hyperplane.

2.8 The k-Nearest Neighbors Classifier

Nearest neighbor classification (Cover and Hart, 1967) is one of the most well-known classification methods. The k-nearest neighbors performs vote on the class of a new sample based on the classes of the k nearest training samples. The k-nearest neighbors (k-NN) algorithm attempts to improve upon the previous rule. It determines the k-nearest training samples to the test sample being classified, and uses these samples to vote on the class label of the test sample. Let n training patterns be denoted as $x^i(l)$, i = 1, 2 , ... n_l, l=1, 2, ..., C, where n_l is the number of training patterns from class ω_l,

[1] The *maximum-likelihood estimate* of **w** is, that value $\hat{\mathbf{w}}$, which maximizes $f(\mathbf{x}/\mathbf{w})$.

$\sum_{l=1}^{C} n_l = n$, and C is the total number of categories. Let $K_l(k,n)$ be the number of patterns from class ω_l among the k-nearest neighbors of pattern **x**. The nearest neighbors are computed from the n-training patterns. The k-NN decision rule $\delta(x)$ is defined as

$$\delta(x)=\omega_j \quad \text{if } K_j(k,n) \geq K_i(k,n) \quad \text{for } j \neq i \tag{2.27}$$

The Euclidean distance metric is commonly used to calculate the k-nearest neighbors. Let $D(x,x^i)$ denote the Euclidean distance between two patterns vectors, **x** and $\mathbf{x}^i$, then

$$D(x,x^i) = \sum_{j=1}^{d}(x_j - x_j^i)^2 = -2M(x,x^i) + \sum_{j=1}^{d} x_j^2$$
where
$$M(x,x^i) = \sum_{j=1}^{d} x_j x_j^i - \frac{1}{2}\sum_{j=1}^{d}(x_j^i)^2, \tag{2.28}$$

and d is the number of features. $M(\mathbf{x},\mathbf{x}^i)$ as the matching score between the test patterns x and the training pattern x^i. So finding the minimum Euclidean distance is equivalent to finding the maximum matching score.

2.9 Difference between Perceptron and Gaussian Classifier

The single layer perceptron is capable of classifying *linearly separable* patterns. The Gaussian distributions of the two patterns in the maximum-likelihood Gaussian classifier do certainly *overlap* each other and therefore not exactly separable; the extent of the overlap is determined by the mean vectors and covariance matrices.
The perceptron convergence algorithm is non-parametric in the sense that it makes no assumptions concerning the form of underlying distributions; it operates by concentrating on errors that occur where the distributions overlap. On the other hand, the maximum-likelihood Gaussian classifier is parametric; it is based on the assumption that the underlying distributions are Gaussian.

The perceptron convergence algorithm is both adaptive and simple to implement; its storage requirement is confined to the set of synaptic weights and threshold. In contrast, the design of the maximum-likelihood Gaussian classifier is fixed; it can be made adaptive, but at the expense of increased storage requirement and more complex computations.

2.10 Conclusion

This chapter has presented an introduction to pattern recognition and a comprehensive overview of supervised classification. Bayesian classification, linear and quadratic discriminate functions, maximum likelihood Gaussian classifiers and the K-nearest neighbours classifier have been discussed. Material presented later in this thesis refers back to this chapter.

Chapter 3

Artificial Neural Networks

3.1 Introduction

The human brain is a highly complex nonlinear information processing system. Parallel processing at all levels of information processing is of major importance to intelligent systems. The recognition of patterns of sensory input is one of the functions of the brain. The question often asked is how do we perceive and recognize faces, objects and scenes. Even in those cases where only noisy representations exist, we are still able to make some inference as to what the pattern represents. Much of the research is motivated by the desire to understand and build parallel neural net classifiers inspired by biological neural networks.

Artificial neural networks have been applied successfully to a variety of tasks. These tasks include pattern recognition, classification, learning and decision making. A neural network derives its computing power through, its massively parallel-distributed structure and its ability to learn and therefore generalize. These two information-processing capabilities make it possible for neural networks to solve complex problems. All knowledge in ANNs is encoded in weights. One possible reason for the good performance of these networks is that the non-linear statistical analysis of the data performed tolerates a considerable amount of imprecise and incomplete input data. Current applications can be viewed as falling into two broad categories. In one class of applications, neural models are used as modelling tools to simulate various neurobiological phenomena. In the second class of applications, neural models are used as computational tools to perform specific information processing tasks.

Both categories use the same basic building blocks, but the final structures and measures of performance are tailored to the end use. For example, neural models have been used for diagnostic problem solving. In a diagnostic problem one is given certain manifestations such as symptoms, signs, and laboratory test abnormalities and must determine the disease causing those findings. Each input node typically represents a different manifestation and each output node represents a different disorder.

By contrast, a neuroscientist may regard back propagation of error as biologically unlikely and could regard the feedforward structure of the multilayer perceptron as being too limited as a model for biological systems. Such a researcher may choose to use a more biologically oriented paradigm, which is able to model specific aspects of biological systems. Thus, although it is possible to define the field of neural networks in terms of systems of parallel non-linear interconnected processing elements, it must be borne in mind that, within the field, there is a very wide variety of approaches and an

intending user will need to carefully assess the most appropriate for the task in hand. There is at present no universally "best" paradigm.

The ability of a neural network to perform or learn a certain task or function very much depends on the properties of the neurons used, the network architecture and the rules for modifying structure or information within the network.

- **Biologically Inspired Neural Networks**

The elementary nerve cell, called a neuron, is the fundamental building block of the biological neural network. A neuron is built up of three parts: the cell body, the dendrites, and the axon as shown in Figure 3-1. The body of the cell contains the nucleus of the neuron and carries out the biochemical transformation necessary to synthesis enzymes and other molecules necessary to the life of the neuron. Each neuron has a hair-like structure of dendrites around it. They branch out into a tree-like from around the cell body. The dendrites are the principal receptors of the neuron and serve to connect its incoming signals. The biological neuron is a complex cell that incorporates both biological and signal processing characteristics.

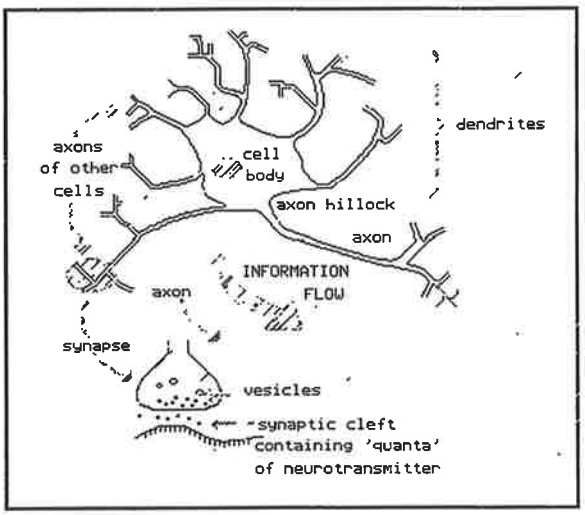
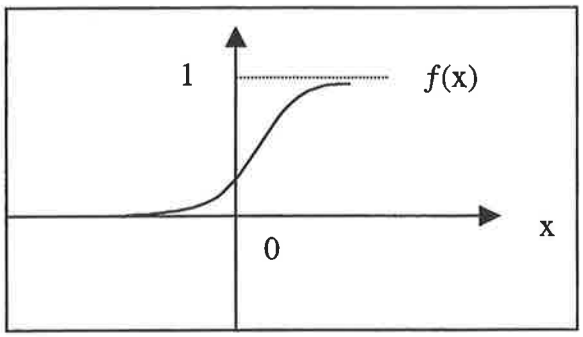


Figure 3-1: A simplified sketch of a biological neuron

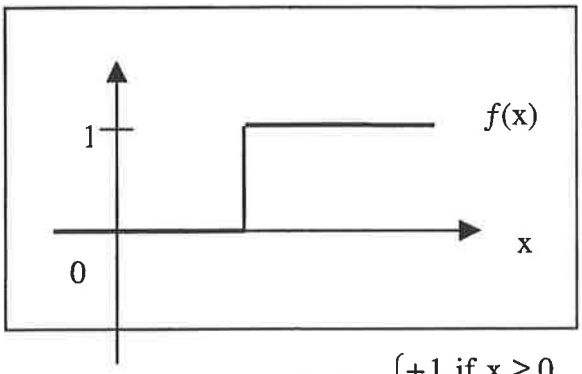
A number of abstract mathematical models have been proposed to capture varying levels of neuronal complexity. The McCulloch-Pitts, (1943) neuron, a summation-threshold device, is perhaps the simplest and most generally used. The sigmoid function is by far the most common form of activation function used in the construction of neural networks. Any function that is monotonically increasing and continuous can be used as an activation function in neuron modelling.

3.2 Types of Activation Function

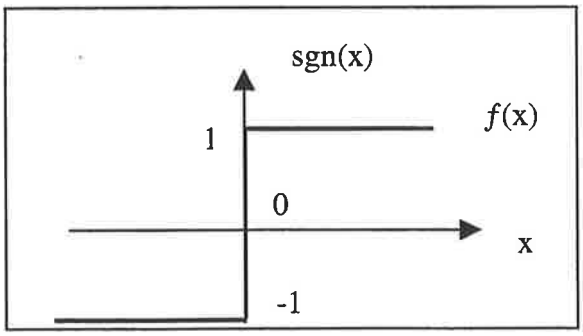
The input-output characteristics of the more general sigmoidal function, threshold function and Sign function are shown in Figure 3-2.



(a): Sigmoid function. $f(x) = \frac{1}{1+e^{-ax}}$



(b): Threshold function. $f(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ 0 \text{ if } x \prec 0 \end{cases}$



(c): Sign function. $sgn(x) = \begin{cases} +1 \text{ if } x \succ 0 \\ -1 \text{ if } x \prec 0 \end{cases}$

Figure 3-2: (a): Sigmoid function. (b): Threshold function. (c): Sign function.

3.3 Learning Techniques

One of most significant attributes of a neural network is its ability to learn by interacting with its environment. This is accomplished through a *learning rule*, by adjusting the weights of the network to improve the performance measure. Learning process can be viewed as search in a multidimensional parameter space for a solution, which gradually optimizes a prespecified objective function.

There are several learning methods such as error-correction learning, Hebbian learning, competitive learning, Boltzmann learning, reinforcement learning.

- **Supervised Learning Versus Unsupervised Learning**

In supervised learning, each input received from the environment is associated with a specific desired target pattern. It is also known as learning with a teacher or associative learning. Usually, the weights are synthesized gradually, and at each step of the learning process they are updated so that the error between the networks output and a corresponding desired target is reduced. Examples of supervised learning algorithms include the least-mean-square algorithm and its generalisation is known as the back-propagation algorithm. The back-propagation algorithm has emerged as the most widely used and successful algorithm for the design of MLP. There are two distinct phases to the operation of the back-propagation learning: the forward phase and the backward phase. In the forward phase the input signals propagate through the network layer by layer, eventually producing some response at the output of the network. The actual response so produced is compared with a desired response, generating error signals that are then propagated in a backward direction through the network. In this backward phase of operation, the free parameters of the network are adjusted so as to minimize the sum of squared errors.

Unsupervised learning involves the clustering or the detection of similarities among unlabeled patterns of a given training set. The idea here is to optimize some criterion function defined in terms of the output activity of the units in the network. Here the weights and the outputs of the network are usually expected to converge to representations that capture the statistical regularities of the input data. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representation for encoding features of the input and thereby new classes automatically. To perform unsupervised learning, for example, we may use competitive learning rule for a two-layer network, input and competitive layer. In its simplest form, the network operates in accordance with a winner-takes-all strategy. The input layer receives the available data. The competitive layer consists of neurons that compete with each other for the opportunity to respond to features contained in the input data.

3.4 The Perceptron

The single-layer perceptron depicted in Figure 3-3 has a single neuron. The term was first used by Frank Rosenblatt (1958). It is the simplest feedforward neural network structure. The single node computes a weighted sum of the input elements, subtracts a threshold and passes the result through a hard limiting nonlinearity such that the output y is either +1 or −1. The decision rule is to respond class A if the output is +1 and class B if the output is −1. The input-output relationship of the unit is represented by the inputs x_i, output y, connection weight w_i, and threshold θ, and as follows:

$$y = \sum_{i=1}^{p} w_i x_i - \theta \tag{3.1}$$

The perceptron learns by adjusting its weight. Connection weights $w_1, w_2, ...w_p$ and the threshold θ in a perceptron can be fixed or adapted using a number of different algorithms. First connection weights and the threshold value are initialized to small random non-zero values. Then a new input with N continuos valued elements are applied to the input and the output is computed. The equation of the boundary line depends on the connection weights and the threshold. For analyzing the behaviour of nets such as the perceptron is to plot a map of the decision regions created in the p-dimensional space spanned by the input variables $x_1, x_2, ...x_p$. In the case of an elementary perceptron, there are two decision regions separated by hyperplane defined by

$$\sum_{i=1}^{p} w_i x_i - \theta = 0 \tag{3.2}$$

These decision regions specify which input values result in class A and which result in class B response. For the case of two input variables the decision boundary takes the form of a straight line. A point that lies above the boundary line is assigned to class A and point that lies below the boundary line is assigned to class B. Connection weights are adapted only when an error occurs. Such a perceptron is limited to performing pattern classification with only two classes. The single layer perceptron can be used with both continuous valued and binary inputs. The externally applied threshold is denoted by θ. The effect of the threshold is to shift the decision boundary away from the origin.

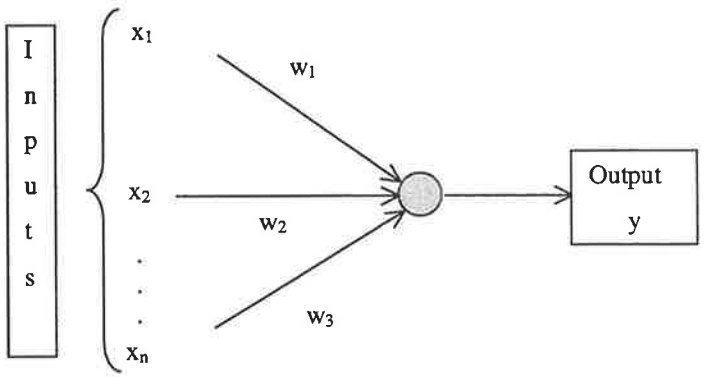


Figure 3-3: Single-layer Perceptron

3.5 Limitations of Single Layer Perceptron

Linear separability limits single-layer networks to classification problems in which the sets of points corresponding to input values can be separated geometrically. For two input case the separator (decision boundary) is a line.

The perceptron forms two decision regions separated by a hyperplane. Rosenblatt proved that if the inputs from two classes are separable then the perceptron convergence procedure converges and positions the decision hyperplane between two classes. This decision boundary separates all samples from the A and B classes. An example of the use of perceptron convergence procedure is presented in Figure 3-4. Circles and crosses represent samples from class A and B.

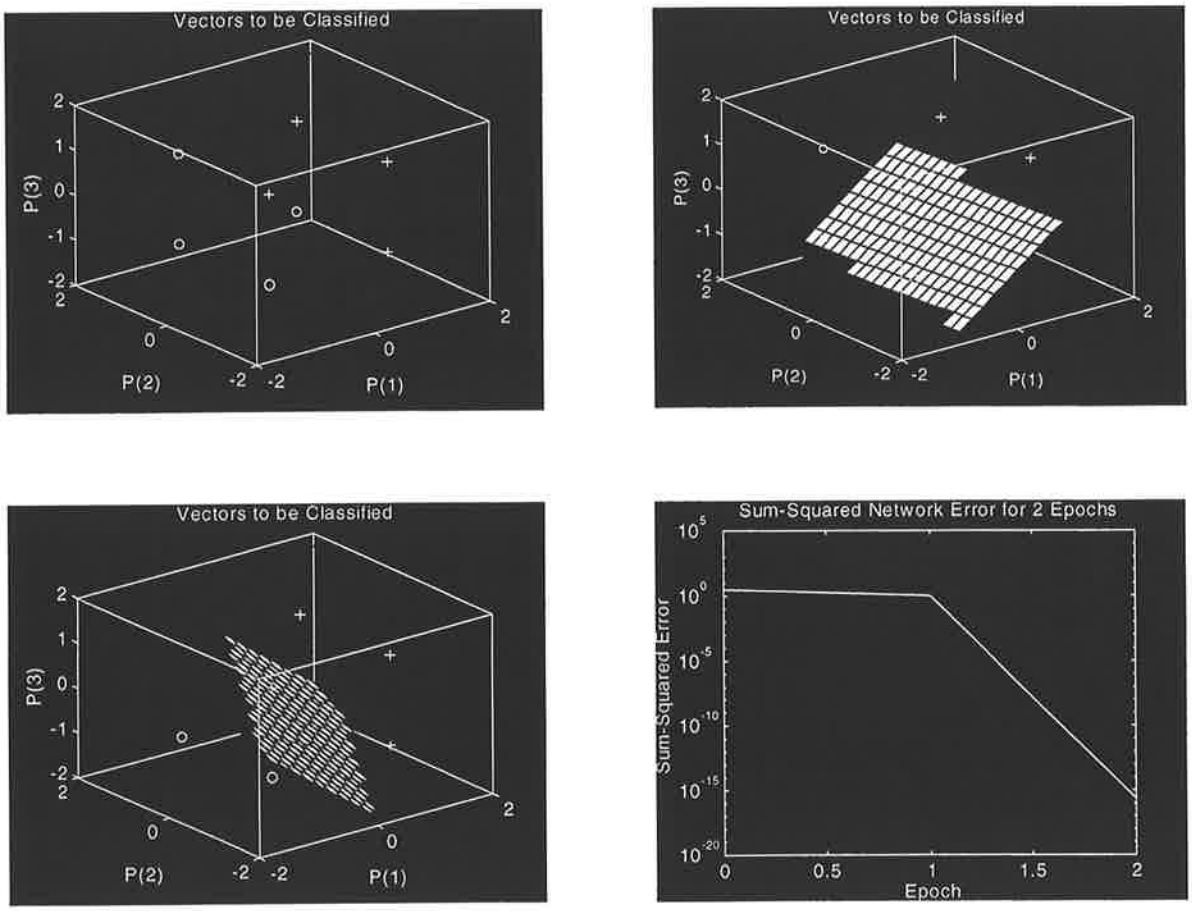


Figure 3-4: Classification with a 3-input perceptron.

1. Initialise weights and thresholds

Define weights w_{it} to be the weight from input i at time t, and θ to be the threshold value in the output node. Set w_0 to be $-\theta$, the bias , and x_0 to be always 1. Set $w_i(0)$ to small random values, thus initialising all the weights and the threshold.

2. Present input and desired output

Calculate actual output

$$y(t) = f_h\left(\sum_{i=0}^{n} w_i(t)x_i(t)\right)$$

3. Adapt weights

if correct	$w_i(t+1) = w_i(t)$
if output 0, should be 1 (class A)	$w_i(t+1) = w_i(t) + x_i(t)$
if output 1, should be 0 (class B)	$w_i(t+1) = w_i(t) - x_i(t)$

The weights are unchanged if the net makes the correct decision. Also, weights are not adjusted on input lines which do not contribute to the incorrect response, since each weight is adjusted by the value of the input on that line, x_i which would be zero.

Figure 3-5: Perceptron Learning Algorithm

3.6 Multilayer Perceptron (MLP)

The Rumelhart-Hinton multilayer perceptron is a feed-forward network with an arbitrary number of layers. A three-layer perceptron with two layers of hidden unit is shown in Figure 3-6. The network is organised into layers with an input layer, an output layer, and hidden layers in between. Usually either one or two hidden layers are used. Each node in the first layer behaves like a single-layer perceptron and has a high output only for points on one side of the hyperplane formed by its weights and offset. The input-output relationship of each unit is represented by the inputs x_i, output y, connection weight w_i, threshold θ, and differentiable function ϕ as follows:

$$y = \phi\left(\sum_{i=1}^{p} w_i x_i - \theta\right)$$

It extends the perceptron principle and capabilities in important ways by including more layers of variable weights and by replacing the hard non-linearity by a smooth sigmoidal function. A single-layer perceptron forms half-plane decision regions. A two layer perceptron can form any, possibly unbounded, convex region in the space spanned by the inputs (Lippmann, 1987). It can generate arbitrarily complex decision regions. These nets can be trained with back-propagation algorithm (BP) also known as *generalised delta rule*. The BP learning procedure, based on the chain rule and gradient descent, provides one of the most efficient learning techniques in this respect.

The back-propagation algorithm uses gradient descent search in the weight space to minimize the error between the target output and the actual output. There are two distinct passes of computations. The first pass is referred to as the forward pass, and the second one as the backward pass.

In the forward pass the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. In the forward phase of computation begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of this layer.

The backward pass, on the other hand, starts at the output layer by passing the error signals leftward through the network, layer by layer, and recursively computing the local gradient for each neuron. This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule of equation (3.14).

For the presentation of each training example the input pattern is fixed throughout the process encompassing the forward process followed by the backward process.

The back propagation algorithm, despite it's simplicity and popularity has several drawbacks. It is slow, and needs thousands of iterations to train a network pertaining to a simple problem. The algorithm is also dependent on the initial weights, and the values of momentum, α and learning rate η.

$$E = \frac{1}{2}\sum_{k}(t_k - y_k)^2 \tag{3.3}$$

where,

t_k = target output of the kth neuron in the output layer

y_k = actual output of the kth neuron in the output layer

The derivative of the error, with respect to each weight is set proportional to weight change as:

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \tag{3.4}$$

where η is called the learning rate. It is a general practice to accelerate the learning procedure by introducing a momentum term α into the learning equation, as follows:

$$\Delta w_{jk}(t+1) = -\eta \frac{\partial E}{\partial w_{jk}}(t+1) + \alpha \Delta w_{jk}(t) \tag{3.5}$$

where,

w_{jk} = weight from the jth unit to the kth unit

There is another form of the weight update rule as given below in (3.6):

$$\Delta w_{jk}(t+1) = -(1-\alpha)\eta \frac{\partial E}{\partial w_{jk}}(t+1) + \alpha \Delta w_{jk}(t) \tag{3.6}$$

The factor $(1-\alpha)$ is included so the learning rate does not need to be stepped down as the constant α is increased.

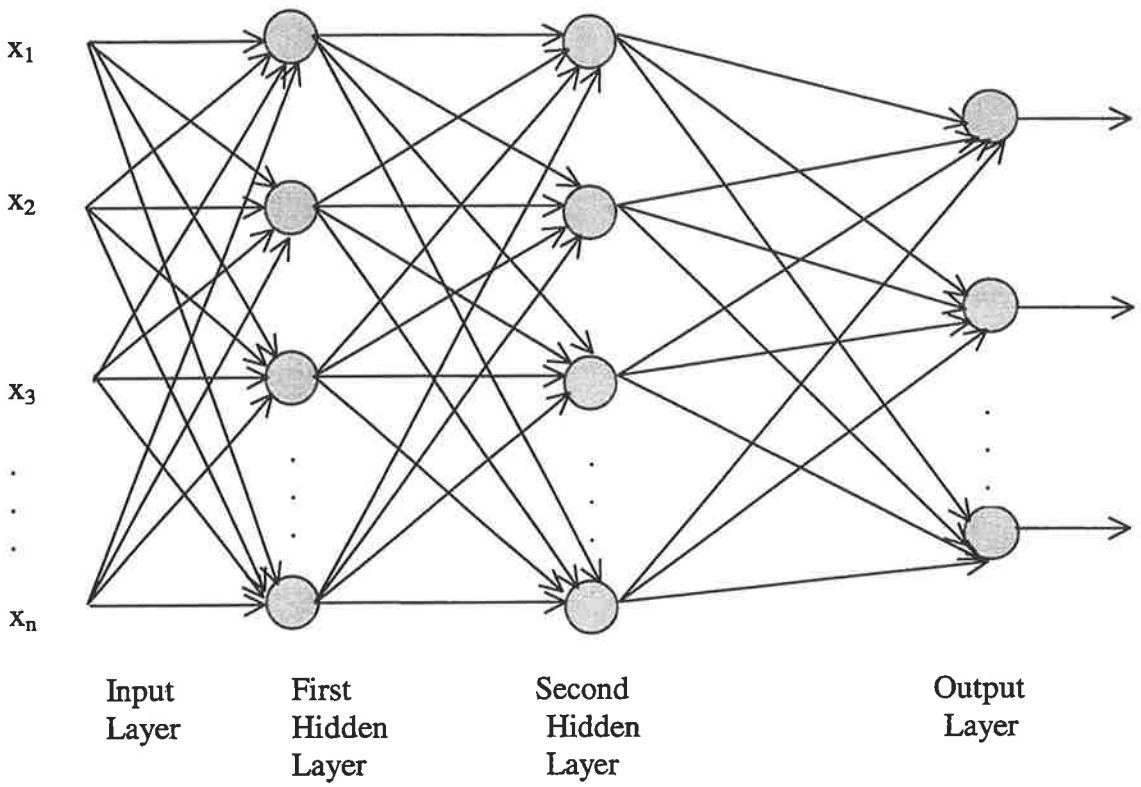


Figure 3-6: The Multi-Layer Perceptron architecture.

3.7 Approximate Realization of Continuous Mappings by Neural Network

Let $\phi(x)$ be a non-constant, bounded and monotonically increasing continuous function. Let K be a compact subset (bounded closed subset) of $\mathbf{R}^n$ and $f(x_1,...,x_n)$ be a real valued continuous function on K. Then for an arbitrary $\epsilon > 0$, there exists an integer N and real constants c_i, θ_i (i = 1, ... , N), w_{ij} (i = 1, ... , N, j = 1, ... , N), such that

$$\bar{f}(x_1,...,x_n) = \sum_{i=1}^{N} c_i \phi\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right)$$

satisfies $\max_{x \in K} \left| f(x_1,...,x_n) - \bar{f}(x_1,...,x_n) \right| < \epsilon$. In other words, for an arbitrary $\epsilon > 0$, there exists a three-layer network whose output functions for the hidden layer are $\phi(x)$, whose output functions for input and output layers are linear and which has an input-output function $\bar{f}(x_1,...,x_n)$ such that

$$\max_{x \in K} \left| f(x_1,...,x_n) - \bar{f}(x_1,...,x_n) \right| < \epsilon .$$

We start with Irie-Miyake's theorem to prove it.

Irie-Miyake's Integral Formula

Let $\psi(x) \in L^1(\mathbf{R})$, that is, let $\psi(x)$ be absolutely integrable and $f(x_1,...,x_n) \in L^2(\mathbf{R}^n)$. Let $\psi(\xi)$ and $F(w_1,\ldots,w_n)$ be Fourier transforms of $\psi(x)$ and $f(x_1,...,x_n)$ respectively.

If $\psi(1) \neq 0$, then

$$f(x_1,...,x_n) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) \frac{1}{(2\pi)^n \psi(1)} F(w_1, \ldots, w_n) e^{(iw_0)} dw_0 dw_1 \ldots dw_n .$$

This formula clearly asserts that if we set

$$I_{\infty,A}(x_1,...,x_n)$$
$$= \int_{-A}^{A} \cdots \int_{-A}^{A} \left[\int_{-\infty}^{\infty} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) \frac{1}{(2\pi)^n \psi(1)} F(w_1, \ldots, w_n) e^{(iw_0)} dw_0 \right] dw_1 \ldots dw_n .$$

Then

$$\lim_{A \to \infty} \left\| I_{\infty,A}(x_1,...,x_n) - f(x_1,x_2,...,x_n) \right\|_{L^2} = 0$$

Irie and Miyake (1988) asserted that a three-layer network can represent arbitrary function with an infinite number of computational units. In this formula w_0 corresponds to threshold, w_i corresponds to connection weights and $\psi(x)$ corresponds to the output function of the units. However, the sigmoid function does not satisfy the condition of

this formula that $\psi(x)$ be absolutely integrable and so the formula does not directly give the realization of the theorem of functions by networks.

Lemma 1

Let $\phi(x)$ be a nonconstant, bounded and monotonically increasing continuous function. For $\alpha > 0$, if we set

$$g(x) = \phi(x + \alpha) - \phi(x - \alpha),$$

Then $g(x) \in L^1(\mathbf{R})$, that is ,

$$\int_{-\infty}^{\infty} |g(x)|\, dx \prec \infty$$

Furthermore, for some $\delta > 0$, if we set

$$g_\delta(x) = \phi(x/\delta + \alpha) - \phi(x/\delta - \alpha),$$

then the result of Fourier transform $G_\delta(\xi)$ of $g_\delta(x)$ at $\xi = 1$ is non-zero.

Proof: Let $|g(x) \leq M|$. For $L > M$,

$$\begin{aligned}\int_{-L}^{L} |g(x)|\, dx &= \int_{-L}^{L} g(x)\, dx \\ &= \int_{-L+\alpha}^{L+\alpha} \phi(x)\, dx \\ &\quad - \int_{-L-\alpha}^{L-\alpha} \phi(x)\, dx \\ &= \int_{L-\alpha}^{L+\alpha} \phi(x)\, dx \\ &\quad - \int_{-L-\alpha}^{-L+\alpha} \phi(x)\, dx \leq 4\alpha M\end{aligned}$$

Therefore,

$$\lim_{L \to \infty} \int_{-\infty}^{\infty} |g(x)|\, dx \prec \infty$$

We show that for some $\delta > 0$, $G_\delta(1) \neq 0$. If the assertion does not hold, then for any $\delta > 0$,

$$\int_{-\infty}^{\infty} ((\phi(x/\delta + \alpha) - \phi(x/\delta - \alpha))e^{-ix} dx = 0$$

By the change of the variable,

$$\int_{-\infty}^{\infty} (\phi(x + \alpha) - \phi(x - \alpha))e^{-ix\delta} dx = 0 \qquad \text{for any } \delta \succ 0 \tag{3.7}$$

Taking the complex conjugate of the equation (3.7)

$$\int_{-\infty}^{\infty} (\phi(x+\alpha) - \phi(x-\alpha))e^{ix\delta}dx = 0 \qquad \text{for any } \delta \succ 0 \tag{3.8}$$

Since the Fourier transform $G_1(\xi)$ of $g_1(x) = \phi(x+\alpha) - \phi(x-\alpha), \in L^1(\mathbf{R})$ is continuous, so from (3.7) and (3.8), $G_1(\xi)$ is identically zero. Therefore,

$$\phi(x+\alpha) - \phi(x-\alpha) \equiv 0$$

This is a contradiction because $\phi(x)$ is not a constant. This lemma holds for $\phi(x)$ which is locally summable.

Lemma 2

Let $A_i > 0$ (i = 1,...m), K be a compact bounded closed subset of $\mathbf{R}^n$ and $h(x_1,...x_m, t_1,...t_n)$ be a continuous function on $[-A_1, A_1] \times ... \times [-A_m, A_m] \times K$.

Then the function defined by the integral

$$H(t) = \int_{-A_1}^{A_1} \cdots \int_{-Am}^{A_m} h(x_1,\ldots,x_m,t_1,\ldots,t_n)\, dx_1 \ldots dx_m$$

can be approximated uniformly on K by the Riemann sum

$$H_N(\mathbf{t}) = \frac{2A_1 \ldots 2A_m}{N^m} \times \sum_{k_1 \ldots k_m}^{N-1} h\left(-A_1 + \frac{k_1.2A_1}{N}, \cdots, -A_m + \frac{k_m.2A_m}{N}, t_1, \ldots t_n\right)$$

In other words, for an arbitrary $\in > 0$, there exists a natural number N_0 such that $N \geq N_0$,

$$\max \left|H(\mathbf{t}) - H_N(\mathbf{t})\right| \prec \in .$$

Proof: The function $h(\mathbf{x},\mathbf{t})$ is continuous on the compact set $[-A_1, A_1] \times \cdots \times [-A_m, A_m] \times K$, so $h(\mathbf{x},\mathbf{t})$ is uniformly continuous. Therefore for any $\in > 0$, we can take the integer N_0 such that if $N \geq N_0$ and

$$\left|x_i - \left(A_i + \frac{k_i.2A_i}{N}\right)\right| \prec \frac{2A_i}{N} (i = 1,\ldots,m) \quad \text{then}$$

$$\left|h(x_1,\ldots,x_m,t_1,\ldots,t_n) - h\left(-A_1 + \frac{k_1.2A_1}{N}, \cdots, -A_m + \frac{k_m.2A_m}{N}, t_1, \ldots t_n\right)\right| \prec \frac{\in}{2A_1 \ldots 2A_m}$$

This proves lemma 2.

Step 1: Because $f(\mathbf{x})$ ($\mathbf{x} = x_1, \ldots, x_n$))is a continuous function on a compact set K of $\mathbf{R}^n$, $f(\mathbf{x})$ can be extended to be a continuous function on $\mathbf{R}^n$ with compact support.
If we operate the mollifier[1] ρ_α^* on $f(\mathbf{x})$, $\rho_\alpha^* f(\mathbf{x})$, is C^∞ -function with compact support. Furthermore, $\rho_\alpha^* f(\mathbf{x}) \rightarrow f(\mathbf{x})$ ($\alpha \rightarrow +0$) uniformly on $\mathbf{R}^n$ Therefore we suppose $f(\mathbf{x})$ is a C^∞ -function with compact support to prove the above theorem. By the Paley-Wiener theorem (Yosida 1968), the Fourier transform $F(\mathbf{w})$ ($\mathbf{w} = w_1, \ldots w_n$) of $f(\mathbf{x})$ is real analytic and, for any integer N, there exists a constant C_N such that

$$|F(\mathbf{w})| \le C_N (1+|w|)^{-N} \tag{3.9}$$

In particular, $F(\mathbf{w}) \in L^1 \cap L^2(\mathbf{R}^n)$.

We define $I_A(x_1, \ldots, x_n)$, $I_{\infty,A}(x_1, \ldots, x_n)$ and $J_A(x_1, \ldots, x_n)$ as follows:

$$I_A(x_1,\ldots,x_n) = \int_{-A}^{A} \cdots \int_{-A}^{A} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) \frac{1}{(2\pi)^n \psi(1)} F(w_1, \ldots, w_n) e^{(iw_0)} dw_0 dw_1 \ldots dw_n. \tag{3.10}$$

$$I_{\infty,A}(x_1,\ldots,x_n) = \int_{-A}^{A} \cdots \int_{-A}^{A} \left[\int_{-\infty}^{\infty} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) \frac{1}{(2\pi)^n \psi(1)} F(w_1, \ldots, w_n) e^{(iw_0)} dw_0 \right] dw_1 \ldots dw_n. \tag{3.11}$$

$$J_A(x_1,\ldots,x_n) = \frac{1}{(2\pi)^n} \int_{-A}^{A} \int_{-A}^{A} F(w_1, \ldots, w_n) e^{\left(i\sum_{i=1}^{n} x_i w_i\right)} dw_1 \ldots dw_n. \tag{3.12}$$

where $\psi(x) \in L^1$ is defined by

$$\psi(x) = \phi(x/\delta + \alpha) - \phi(x/\delta - \alpha),$$

for some α and δ so that $\psi(x)$ satisfies Lemma 1.

The following equality is the essential part of the proof of I-M integral formula

$$I_{\infty,A}(x_1,\ldots,x_n) = J_A(x_1,\ldots,x_n) \tag{3.13}$$

and this is derived from

$$\int_{-\infty}^{\infty} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) e^{(iw_0)} dw_0 = e^{\left(i\sum_{i=1}^{n} x_i w_i\right)} \cdot \psi(1) \tag{3.14}$$

Using the estimate of $F(\mathbf{w})$ we can prove

[1] The operator ρ_α^* is called a mollifier

$$\lim_{A \to \infty} J_A(x_1,...,x_n) = f(x_1,...,x_n)$$

uniformly on $\mathbf{R}^n$. Therefore

$$\lim_{A \to \infty} I_{\infty,A}(x_1,...,x_n) = f(x_1,...,x_n)$$

uniformly on $\mathbf{R}^n$. That is we can state that for any $\epsilon > 0$, there exits $A > 0$ such that

$$\max_{x \in \mathbf{R}^n} \left| I_{\infty,A}(x_1,...,x_n) - f(x_1,...,x_n) \right| < \frac{\epsilon}{2} \qquad (\dagger)$$

Step 2: We will approximate $I_{\infty,A}$ by finite integrals on K. For any $\epsilon > 0$, fix A which satisfies (†).

For $A' > 0$, set

$$I_{A',A}(x_1,...,x_n) = \int_{-A}^{A} \cdots \int_{-A'}^{A'} \left[\int_{-A'}^{A'} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0 \right) \frac{1}{(2\pi)^n \psi(1)} F(w_1, \ldots, w_n) e^{(iw_0)} dw_0 \right] dw_1 \ldots dw_n.$$

We will show that, for any $\epsilon > 0$, we can take $A' > 0$ so that

$$\max_{x \in K} \left| I_{A',A}(x_1,...,x_n) - I_{\infty,A}(x_1,...,x_n) \right| < \frac{\epsilon}{2} \qquad (\ddagger)$$

Using the following equation

$$\int_{-A'}^{A'} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0 \right) e^{(iw_0)} dw_0$$
$$= \int_{\sum_{i=1}^{n}(x_i w_i) - A'}^{\sum_{i=1}^{n}(x_i w_i) + A} \psi(t) e^{(-it)} dt . e^{\left(i \sum_{i=1}^{n} x_i w_i \right)}$$

the fact $F(\mathbf{x}) \in L^1$ and compactness of $[-A, A]^n \times K$ we can take A' so that

$$\left| \int_{-A'}^{A'} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0 \right) e^{(iw_0)} dw_0 - \int_{-\infty}^{\infty} \psi\left(\sum_{i=1}^{n} x_i w_i - w_0 \right) e^{(iw_0)} dw_0 \right|$$
$$\leq \frac{\epsilon (2\pi)^n |\psi(1)|}{\left(2 \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} |F(x)| dx + 1 \right)} \int_{-A'}^{A'} \cdots \int_{-A'}^{A'} |F(x)| dx$$

on K. Therefore

$$\max_{x \in K} \left| I_{A',A}(x_1,...,x_n) - I_{\infty,A}(x_1,...,x_n) \right| \leq \frac{\epsilon}{\left(2 \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} |F(x)| dx + 1 \right)} \int_{-A'}^{A'} \cdots \int_{-A'}^{A'} |F(x)| dx < \frac{\epsilon}{2}$$

Step 3: From (†) and (‡) we can say that for any $\epsilon > 0$, there exits A, $A' > 0$ such that

$$\max_{x\in K}\left|f(x_1,...,x_n)-I_{A',A}(x_1,...,x_n)\right|\prec\in \qquad (*)$$

that is to say $f(\mathbf{x})$ can be approximated by the finite integral $I_{A',A}(\mathbf{x})$ uniformly on K. The integrand of $I_{A',A}(\mathbf{x})$ can be replaced by the real part and is continuous on [-A′, A′] ×...×[-A A] × K, so by Lemma 2 , $I_{A',A}(\mathbf{x})$ can be approximated by the Riemann sum uniformly on K.
Since

$$\psi\left(\sum_{i=1}^{n}x_iw_i-w_0\right)=\phi\left(\sum_{i=1}^{n}w_ix_i/\delta-w_0+\alpha\right)-\phi\left(\sum_{i=1}^{n}w_ix_i/\delta-w_0-\alpha\right),$$

A three-layer network can represent the Riemann sum. Therefore $f(x)$ can be represented approximately by the three-layer networks.

3.8 Derivation of Back-propagation Algorithm

The back-propagation algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent.

Let E_p is the error function for pattern p, t_{pj} represents the target output for pattern p on node j, whilest o_{pj} represents the actual output at the node. w_{ij} is the weight from node i to node j. We begin by defining the cost function as the instantaneous sum of squared difference between the actual output and the desired output is:

$$E_p=\frac{1}{2}\sum_k\left(t_{pj}-o_{pj}\right)^2 \qquad (3.15)$$

The activation of each unit j, for pattern p, can be written as the weighted sum, as in the single layer perceptron.

$$net_{pj}=\sum_i w_{ij}o_{pi} \qquad (3.16)$$

The output from each unit j is the threshold function f_j acting on the weighted sum. In the multilayer perceptron, it is usually the sigmoid function, although it can be any continuously differentiable monotonic function.

$$o_{pj}=f_j\left(net_{pj}\right) \qquad (3.17)$$

By chain Rule, we can write,

$$\frac{\partial E_p}{\partial w_{ij}}=\frac{\partial E_p}{\partial net_{pj}}\frac{\partial net_{pj}}{\partial w_{ij}} \qquad (3.18)$$

Looking at the second term in (3.18) and (3.16)

$$\frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} o_{pk}$$

$$= \sum_k \frac{\partial w_{jk}}{\partial w_{ij}} o_{pk}$$

$$= o_{pi} \tag{3.19}$$

Since $\frac{\partial w_{kj}}{\partial w_{ij}} = 0$ except when k = i when it equals 1. We can define change in error as a function of the change in the net inputs to a unit as

$$-\frac{\partial E_p}{\partial net_{pj}} = \delta_{pj} \tag{3.20}$$

and so (3.18) becomes

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} o_{pi} \tag{3.21}$$

Decreasing the value of E_p therefore means making the weight changes proportional to $\delta_{pj} o_{pi}$, i.e.

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi} \tag{3.22}$$

We can now find δ_{pj} for each of the unit. Using (3.6) and the chain rule, we can write

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}} \tag{3.23}$$

Consider the second term, and from (3.9),

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f_j'(net_{pj}) \tag{3.24}$$

Consider now the first term in (3.15). From (3.7), we can differentiate E_p with respect to o_{pj}, giving

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) \tag{3.25}$$

Thus

$$\delta_{pj} = f_j'(net_{pj})(t_{pj} - o_{pj}) \tag{3.26}$$

This is useful for the output units, since the target and output are both available but not for the hidden units, since their targets are not known. So if unit j is not an output unit, we can write, by chain rule again. that

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}}$$

$$= \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} \sum_i w_{ik} o_{pi} \tag{3.27}$$

$$= -\sum_k \delta_{pk} w_{jk} \tag{3.28}$$

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{jk} \tag{3.29}$$

This equation represents the change in the error function with respect to the weights in the network. This provides a method for changing the error function so as to reduce the error. The function is proportional to the errors δ_{pk} in subsequent units, so the error has to be calculated in the output unit first and then passed back through the net to the earlier units to allow them to alter their connection weights. Equation (3.26) and (3.29) together define the weight adjustment of output neuron of MLP.

The computation of the local gradient for each neuron of MLP requires knowledge of the derivative of the activation function. Differentiability is only requirement that an activation function would have to satisfy. The sigmoid function is most commonly used as a non-linear continuously differentiable nonlinear activation function. Hyperbolic tangent is another example of sigmoidal nonlinearity. The derivative is simple function of the outputs. Given the output of unit, o_{pj} is given by

$$o_{pj} = f(net) = \frac{1}{(1 + e^{-k\,net})}$$

the derivative with respect to that unit, $f'(net)$, is given by

$$f'(net) = \frac{k e^{-k\,net}}{(1 + e^{-k\,net})^2}$$

$$= k\, f(net)(1 - f(net))$$

$$= k\, o_{pj}(1 - o_{pj})$$

1. Initialise weights and thresholds. Set all weights and thresholds to small random values

2. Present input $x_1, x_2, \ldots x_p$ and desired output

3. Calculate actual output

Each layer Calculates

$$y_{pj} = f\left(\sum_{i=0}^{n-1} w_i x_i\right)$$

and passes that as input to the next layer. The final layer outputs value o_{pj}.

4. Adapt weights

Starting from the output layer, and working backwards.

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_{pj} o_{pj}$$

w_{ij} represents the weights from node i to node j at time t, η is a learning rate, and δ_{pj} is an error term for pattern p on node j.

For output units

$$\delta_{pj} = k o_{pj}(1 - o_{pj})(t_{pj} - o_{pj})$$

For hidden units

$$\delta_{pj} = k o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{jk}$$

where the sum is over the k nodes in the layer above node j.

Figure 3-7: MLP Learning Algorithm

3.9 Practical Issues

There are several practical considerations that must be addressed to successfully use the MLP.

- **Local Optima**

In practice it is highly unlikely that the error function will contain a single optimum; rather the function that is being optimised will tend to be very rough and noisy, and dotted with local optima (Widrow and Lehr, 1990). Since back-propagation training is a gradient descent method, convergence to a local optimum is final and the network becomes trapped at this suboptimal point. Therefore two networks with the same architecture can often achieve the same goal using different sets of weighs. It is sensible to train the network several times, starting with different initial weights for each training instance. For each training run, the weight vector may start in a different basin

of attraction, and hopefully the basin of the global optimum will eventually be encountered. Figure 3-8 shows error surface with a contour plot underneath. Error surface calculates errors for a neuron with a range of possible weight and bias values. Notice the low error point near the middle, and the two valleys leading away from it.

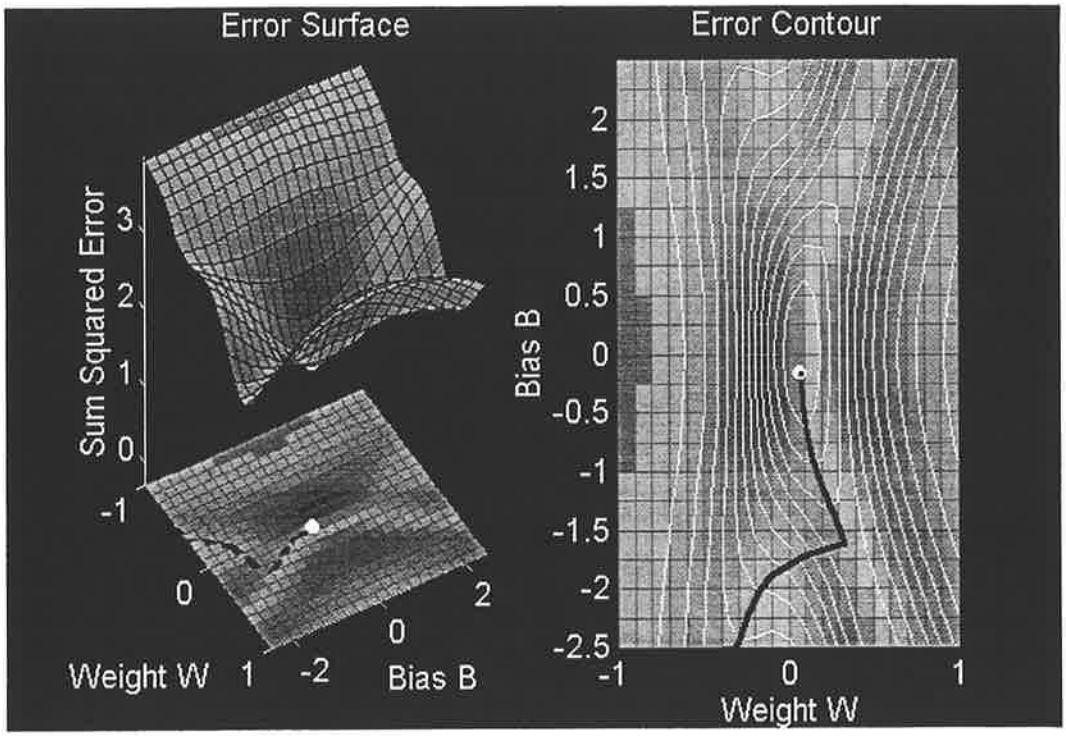


Figure 3-8: The error surface and contour

One method for escaping small local optima is the use of a *momentum* term in training, although this adaptive step-size update method was explicitly developed to speed up training. Momentum allows the step-size parameter to adapt to the local landscape: in regions where successive updates involve gradients with the same sign, the weight update increases in magnitude, metaphorically gaining momentum. In the case where gradients change sign on successive updates, the weight vector has overshot a local minimum and the weight update magnitude decreases to fit in the smaller basin. For an appropriately chosen α, the weight vector can overshoot and thus escape relatively small local optima.

- **Architecture**

The first decision that must be made is the choice of architecture. If too, many hidden neurons are used, the network tends to over-fit the data; if too few, the network may under-fit since it does not posses enough free parameters to perform the mapping. In addition, the number of layers may make a difference to the performance of the MLP. We could go as far as to allow arbitrary forward connections among the neurons. These factors depend on the underlying complexity of the data which is generally unknown, so far near-optimal results a search must be made over the set of possible architectures.

There is a significant amount of research into the use of evolutionary algorithms for this purpose, the different methods varying in the extent of the search. For instance, genetic algorithms have been used to find the best neural connections for a fixed number of hidden nodes, while more extreme approaches have used genetic programming to find the number of hidden layers, hidden nodes, and the interconnections between them.

3.10 Conclusion

This chapter has presented an overview of the perceptron and MLP. A brief review of the biological inspiration for neural networks has been given. The relative merits of supervised and unsupervised learning have been discusssed. The single layer perceptron has been described and its limitation to input values which can be seperated geometrically has been identified. Generalisations to a multilayer perceptron in order to remove this limitation has been outlined and the mathematical theory of MLP with practical issues has been presented.

Chapter 4

Genetic Algorithms

4.1 Introduction

The Genetic Algorithm (GA) were introduced by John Holland (Holland 1995, first published 1975), and a seminal treatment has been given by Goldberg (Goldberg, 1989). Genetic algorithms like neural networks are a biologically motivated paradigm based on natural selection and genetics. They are based on the genetic processes of biological organisms. These algorithms represent the complex structure of a problem by a simple code of bit strings, which mimic the genes in a chromosome. Over many generations natural populations evolve according to the principles of natural selection and *Species*. GA's work with a population of individuals each representing a possible solution to a given problem. Each individual is assigned a fitness score according to how good a solution to the problem it is. The technique is robust and can deal with a wide range of problems. In his recent book, David Goldberg describes genetic algorithms '... search algorithms based on the mechanics of natural selection and natural genetics (resulting in) a search algorithm with some of the innovative flair of human search.'

Genetic algorithms differ from traditional search algorithms as follows.

1. GAs work with a coding of the parameter set, not the parameters themselves.
2. GAs search from a population of points, not from a single point.
3. GAs use pay-off (objective function) information, not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules, not deterministic rules.

The parameters of a problem are usually coded into a string of binary features analogous with chromosomes in biology. This coding is done by the user of the GA. The GA itself has no knowledge at all of the meaning of the coded string. If the problem has more than one the string contains multiple sub-strings or genes, one for each of the parameters. Each coded string represents a possible solution to the problem. The GA works by manipulating a population of such possible coded solutions in a reproduction process driven by a number of genetic operators.

During the reproduction process, new solutions are created by selecting and recombining existing solutions based on pay-off information(often called fitness) using the genetic operators. The process can be compared with natural selection and the Darwinian theory of evolution in biology: fit organisms are more likely to stay alive and reproduce than non-fit organisms. The basic outline of a genetic algorithm is as follows:

```
/*Genetic Algorithm*/

BEGIN
Generate initial population
Determine fitness of each individual

WHILE NOT finished DO
Begin

    FOR population size DO
    BEGIN
        Select two individuals from old generation for mating
        Recombine the two individuals to give two offspring
        Determine fitness of the two offspring
        Insert offspring in new generation
    END
    IF population has converged THEN
        Finished = TRUE
    END
END
```

Figure 4-1: The Basic Genetic Algorithm

4.2 Fitness function

The fitness function reflects the ability of the individual which that chromosomes represents. For each problem, a fitness function has to be solved. For a particular chromosome, the fitness function calculates a single numerical fitness or figure of merit. In genetic terms, the set of parameters represented by a particular chromosome is referred to as a *genotype*. The genotype contains the information required to construct an organism which is referred to as the *phenotype*. The fitness of an individual depends on the performance of the phenotype. The fitness of a string (or solution) can be evaluated in many different ways. If the problem, for example, is finding the root of a mathematical function, the fitness can be the inverse of the square of the function value of the proposed solution. If the problem is finding an optimal neural net, the fitness could be the inverse of the convergence time and zero if the network couldn't learn the problem. It could also be the inverse of the error at the output nodes. The GA is not aware of the meaning of the fitness value, just the value itself. This implies that the GA can't use any auxiliary knowledge about the problem. Starting with a population of random strings, each new population (generated by means of reproduction) is based upon (and replaces) the previous generation. This should, in time, lead to a higher overall fitness, and thus to better solutions to the original problem.

The four most commonly used genetic operators used are selection, crossover, inversion and mutation. With each of these of the operators, only random number generating, string copying and changing of bits are involved. Crossover, mutation and inversion are all applied with a certain probability: for each application of an operator it must be decided whether to apply the operator or not. Selection alone is usually not

enough for the GA to work, so, one or more of the other genetic operators have to be applied to the selected string. Table 4-1 shows sample population of 10 strings.

Table 4-1: Sample Population of 10 strings

	Fitness	Bitstring		Fitness	Bitstring
1	1	1111111111	6	5	1101110101
2	2	1100001000	7	5	0001100010
3	3	0000000001	8	6	1000000001
4	3	1111111000	9	7	0000100010
5	4	0001000100	10	9	1100010101

4.3 Genetic Operators

- **Selection**

Selection is used to choose strings from the population for reproduction. On each generation parents are selected to produce new children. The selection of parents is biased by fitness, so that fit parents produce more children and very unfit solutions produce no children. This is known as *selection.* The genes of good solutions thus begin to proliferate through the population. The chance of selection as a parent is proportionate to chromosomes normalised fitness. In parallel with the natural selection mechanism, strings or solutions with a high fitness are more likely to be selected than less fit strings. The two selection methods applied in this research are described respectively by Goldberg and Whitley. The roulette wheel is shown in Figure 4-2.

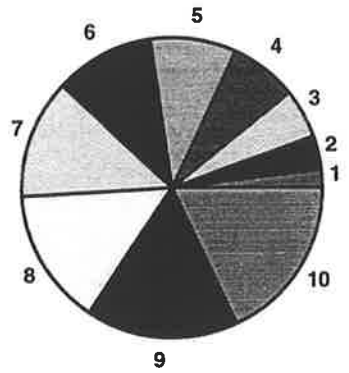


Figure 4-2: The roulette wheel.

With roulette wheel selection, strings are selected with a probability proportional to their fitness. Another method is called rank based selection, where the chance of being selected is defined as a linear function of the rank of an individual in the population. The population must remain sorted by fitness for this method to work. One advantage of rank based selection is that it does not need the fitness scaling necessary with other methods, to prevent high fitness strings from dominating the population, which may result in a premature convergence into a non-optimal solution.

- **Crossover**

The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. Having selected two parents, their chromosomes are recombined by means of crossover. The crossover operator creates new members for the population by combining different parts from two-selected parent strings. First a number of crossover points are chosen at random. A new string is created by using alternate parts of the parent strings. The following various crossover techniques are supported often involving more than one cut point.

one-point crossover: here two chromosomes are joined at a single point and swap ends.

two-point crossover: here two chromosomes are joined at two points and swap the middle section between the two.

uniform crossover: In this the two randomly swap any number of genes that is they are completely reshuffled.

n-point crossover: here the user specifies how many crossover points should be used in this version. Figure 4-3 shows the various crossover methods.

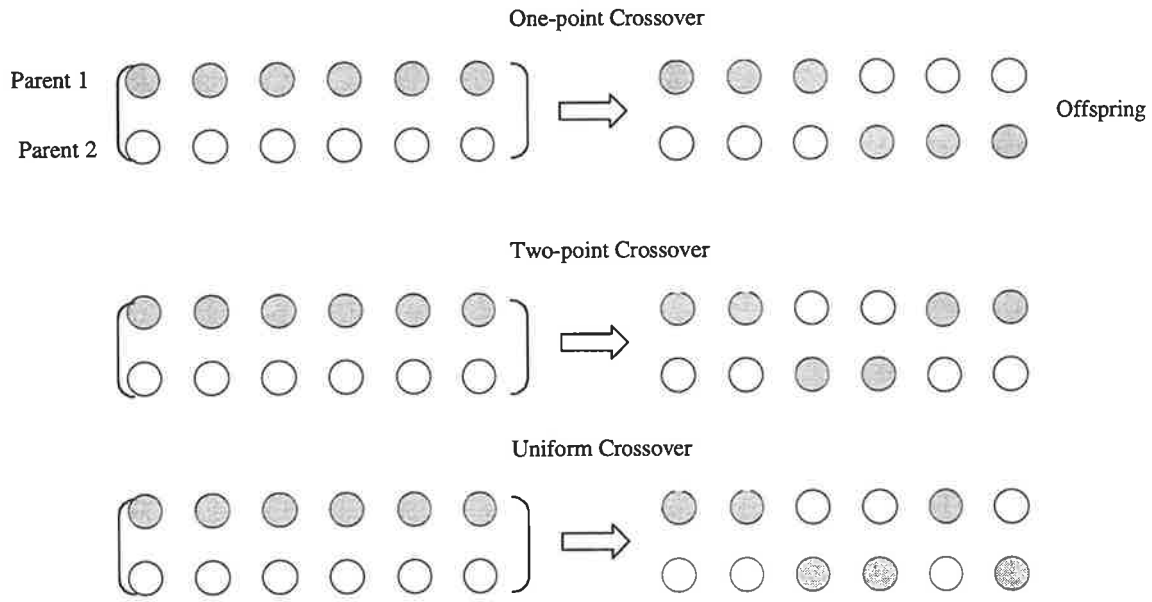


Figure 4-3: Example of 1-point, 2- point and uniform Crossover.

- **Mutation**

Mutation is possibly the simplest of the genetic operators. It randomly flips bits in the string from 0 to 1 or from 1 to 0. The purpose of this string mutation is to improve the algorithm by introducing new solutions not present in the population and by protecting the algorithm against accidental, irrecoverable loss of (valuable) information due for example, to unfortunate crossovers. An example of mutation is shown Figure 4-4.

In order to keep the algorithm from becoming a simple random search, mutation rate has to be low, so it doesn't interfere too much with crossover and inversion. There are some applications however, where selection and mutation are enough for the GA to function.

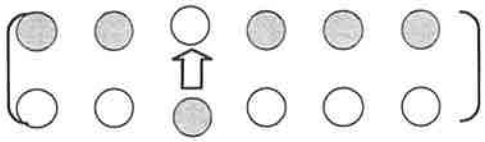


Figure 4-4: Example of mutation; bit 3 has been mutated.

4.4 Mathematical Foundations of Genetic Algorithms

- **The Building Block Hypothesis**

Holland introduced the schema Theorem, which is often viewed as the fundamental theoretical foundation of genetic algorithms. It can be applied to chromosomes that are fixed length strings only.

A *schema* is a template describing a subset of strings with similarities at certain string positions. If we take for example a population of binary strings, schemata for these strings are string themselves, consisting of 0's, 1's and * symbols. The * (wild card symbol) matches either a 0 or a 1. A schema matches a particular string if at every position a 1 in the string and a 0 matches a 0 in the string. If we take strings of length 8 (binary representation of the numbers 0 to 255) the schema 1*0001*1 matches four strings: 10000100, 10000110, 11000100 and 11000110.

Let us take a look at the number of schemata involved in a population of n strings of length l. If each string is built from k symbols (k=2 for binary strings), these are $(k+1)^l$ different schemata (because each of the l positions can be one of k symbols, or an asterisk). So, for our example, there are 256 (2^8) different strings, but there are $(2+1)^8 = 6561$ different schemata. Also a string of length 8 belongs to 2^8 different schemata because each position may take on its actual value or a wild card symbol. For string of length l, this number is 2^l. So, for a population of size n, the population contains somewhere between 2^l and $n.2^l$ schemata. So even moderately sized populations contain a lot of information about important similarities. By using schemata in the genetic search, the amount of information can be much larger than by looking at the strings only. Each schema can be assigned a fitness. This is the average fitness of the members in the population corresponding to that particular schema. We will denote this average fitness with f_S. The *defining length* of a schema, is the distance between the first and the last fixed bits in the schema (non-wild card). Schemata

provide a means for combining attributes, and for analysing their contribution to performance through the population. The order of a schema is the number of non-wild symbols it contains. Looking at the defining length, we note that crossover has a tendency to cut schema of long defining length when the crossover points are chosen uniformly at random: for example, the schema 1*****10 has a higher chance of being cut than *****10* (6/7 or 86% vs. 1/7 or 14%)

A lower bound on the crossover survival probability p_S for a schema with defining length delta can be expressed with the following formula (for crossover with one crossover point):

$$p_s \geq 1 - \frac{\delta}{l-1} p_c \tag{4.1}$$

where p_C is the probability with which crossover will occur, delta is the defining length of the schema, and l is the length of the schema. The formula contains an inequality sign instead of equality sign because even when the schema is cut it can survive if the crossover results in a string that still contains the schema. New strings with the schema can also come into existence.

We can also calculate the effect of selection on the number of schemata. When we have m(t) examples of a particular schema at time t in our population, we can expect

$$m(t+1) = m(t)\, n \frac{f_s}{\sum_{i=1}^{n} f_i} \tag{4.2}$$

examples at time t+1 where n is the population size , f_S the average fitness of the strings representing the schema and $\sum_{i=1}^{n} f_i$ the total fitness of the population. If we rewrite the formula using

$$f_{avg} = \frac{1}{n} \sum_{i=1}^{n} f_i \tag{4.3}$$

for the average fitness of the whole population, it becomes:

$$m(t+1) = m(t) \frac{f_s}{f_{avg}} \tag{4.4}$$

Or: a particular schema grows as the ratio of the average fitness of the schema and the average fitness of the population. So schemata with fitness values above the average population fitness have a higher chance of being reproduced and receive an increasing (exponential) number of samples in the new population. This is carried out for each schema in the population in parallel.

Because mutation has only a very small effect on the number of schemata mutation rate is usually chosen very low), the combine effect of selection and crossover can be expressed with the following formula, which is the result of combining (4.1) and (4.2):

$$m(t+1) \geq m(t) \frac{f_s}{f_{avg}} \left[1 - p_c \frac{\delta}{l-1} \right] \tag{4.5}$$

So a particular schema grows or decays depending upon a multiplication factor. With both selection and crossover the factor depends upon whether the schema's fitness is above or below the population's average fitness and on the length of the schema. Especially, schemata with high fitness and short defining length are propagated exponentially throughout the population. Those short schemata are called building blocks. Crossover directs the genetic search towards finding building blocks (or partial solutions) and also combines them into better overall solutions. Inversion also facilitates the formation of building blocks. Complex problems often consist of multiple parameters which are coded by different genes on the chromosome. With these multiple parameter problems however, complex relations may exist between different parameters. When defining the coding of such a problem, related genes should be positioned close together. When not much is known about the relations between the parameters, inversion can be used as an automatic reordering operator.

- **Implicit parallelism**

The exponential propagation of high fit, small size schemata goes on in parallel, without any more special bookkeeping or memory than a population of n strings. Goldberg presents a more precise count of how many schemata are processed usefully in each generation.

- **Epistasis**

Epistasis is the interaction between different genes in a chromosome. It is the extent to which the contribution of fitness of one gene depends on the values of other genes. The degree of interaction will be different for each gene in a chromosome. A small change to one gene makes a change in resultant chromosome fitness. The resultant change may vary according to the values of other genes.

4.5 Diversity and Convergence

Although convergence to the optimal solution is often used as a measure for an algorithms performance, this criterion has been rejected by Holland (Holland, 1995) according to the argument that even enumerative search converges under this criterion. Rather, the best solution must be found in a reasonable time. Convergence in the genetic algorithms typically refers to the situation that the population becomes homogeneous, containing M copies of the same individual. Further search points can not be reached through crossover, since crossing over identical string results in identical offspring , and the very low mutation rate is the only chance of introducing new genetic material. One hopes that the algorithm has converged upon the optimal

solution; if not, the only recourse is to start the algorithm with a different initial population.

Under this definition of convergence, the diversity of genetic structures in the population is expected to decrease as evolution progresses. Due to the geometric rate at which highly fit individuals propagate into future generations, the GA can converge too quickly without having explored enough of the search space to encounter a global or near global optimum. This phenomenon is called *premature convergence*. In the presence of bit mutation, premature convergence is stagnation in the search for an undetermined amount of time.

The way to stop the GA from converging prematurely is to promote diversity in the population. There are several ways to achieve this:

- Use a high mutation rate.
- Disallow *genetic duplicates* in the population, where two individuals are genotypic duplicates if they are exactly the same. Note the distinction from *phenotypic duplicates*, which are individuals with different strings resulting in the same behaviour.
- Use selection pressure.
- Use a population model that promotes diversity.

A gene is said to have converged when 95% of the population share the same value. The population is said to have converged when all of the genes have converged. Once the population has converged, the ability of the GA to continue to search for better solutions is effectively eliminated. A problem with GAs is that the genes from a few comparatively highly fit individuals may rapidly come to dominate the population, causing it to converge on a local maxima.

4.6 Comparing GA with Back-Propagation

Back-propagation and genetic algorithms are two techniques for optimization and learning, each with its own strengths and weaknesses. The back-propagation learning algorithm is a well-known training method for feedforward neural networks. Back-propagation is a method for calculating the gradient of the error with respect to the weights for a given input by propagating error backward through the network. It generally uses a least-squares optimality criterion. It can find a good set of weights and biases in a reasonable amount of time and sensitive to parameters such as learning rate and momentum. Successful though it is, the algorithm does have some shortcomings. Since it is gradient descent method it has as a tendency to get stuck in local minima of the error surface and thus not find the global minima. Also, it can not handle discontinuous node transfer functions because to compute a gradient requires differentiability.

The basic difference between back-propagation and GA based training mechanisms is that, unlike BP, GA does not make use of local knowledge of the parameter space. Genetic algorithms are good at exploring a large and complex space to find values close to the global optimum. Back-propagation works well on simple training problems. However, as the complexity increases the performance of back-propagation falls off rapidly. The problem of GAs is that they are inherently slow and takes a long time to converge. A hybrid of genetic and back-propagation algorithms which should always find the correct global minima without getting stuck at local

minima. A hybrid training algorithm making use of both GA and BP methods may thus be very useful.

Genetic algorithms should not have the same problem with scaling as back-propagation because they generally improve the current best candidate monotonically. They do this by keeping the current best individual as part of their population while they search for better candidates. Secondly, they have the ability to escape local minima. The mutation and crossover operators can step from a valley across a hill to an even lower valley in the energy function landscape. Control parameters required for standard GA are shown in Table 4-2.

Table 4-2: Control parameters required for standard GA

population size
max. number of generations
probability of crossover
probability of mutation
probability of reproduction
selection scheme
fitness scaling scheme
elitist strategy

4.7 Conclusion

This chapter has presented a brief introduction to genetic algorithms and their mathematical foundations. The fitness function has been introduced and the effects of genetic operators have been considered. Detailed discussion of the four most common operators: selection, crossover, inversion and mutation has been entered into. Description of the mathematical foundations of genetic algorithms has included the building block hypothesis, implicit parallelism and epistasis. Diversity and convergence issues have been considered. Comparison with back-propagation is also presented. A hybrid learning methodology that integrates genetic algorithms and neural networks is presented in chapter 7.

Chapter 5

The Fuzzy set Theory

5.1 Introduction

Fuzzy sets were first introduced by Zadeh in 1965 for handling uncertain and imprecise knowledge in real world applications. It provides effective tools to handle and manipulate imprecise and noisy data. Fuzzy sets can be considered as a generalisation of classical set theory. A classical set (crisp) is normally defined as a collection of elements or objects which can be finite, countable, or overcountable. Each single element can either belongs to or does not belong to the set. Such a classical set can be described in different ways: one can either enumerate the elements that belong to the set describe the set analytically, for instance, by stating conditions for membership or define the member elements by using the *characteristic function*, in which 1 indicates membership and 0 indicates non-membership.

$$\mu_A(x) = \begin{cases} 1 & \text{if and only if } x \in A \\ 0 & \text{if and only if } x \notin A \end{cases}$$

The boundary of the set A is rigid and sharp. On the other hand a fuzzy set is a collection of distinct elements with a varying degree of relevance. For a fuzzy set, the characteristic function allows various degrees of membership for elements of a given set. In fuzzy set, the transition from membership to non-membership function is gradual rather than abrupt. The degree of membership usually denoted by the Greek letter μ. The utility of fuzzy sets lies in their ability to model the uncertain or ambiguous data so often encountered in real life.

5.2 Fuzzy Sets

A fuzzy set is completely characterized by an ordered set of pairs, the first element of which denotes the element and the second the degree of membership.

$$A = \{(x, \mu_A(x), x \in X\} \tag{5.1}$$

$\mu_A(x)$ is called the membership function or grade of membership function. (Sometime it is also referred as the degree of compatibility or degree of truth) of x in A which maps X to the membership space M. When M contains only the two points 0 and 1, A is non-fuzzy and $\mu_A(x)$ is identical to the characteristic function of a non-fuzzy-set. The set

{0,1} is called a valuation set. If the valuation set is allowed to be the real interval [0,1], A is called a fuzzy set. The closer the value of $\mu_A(x)$ is to 1, the more x belongs to A. Clearly, A is a subset of X that has no sharp boundary. The range of the membership function is a subset of the non-negative real numbers whose supremum is finite. Elements with a zero degree of membership are not normally not listed. A fuzzy set is solely represented by stating its membership function. Membership functions for a crisp set and fuzzy set are shown Figure 5-1.

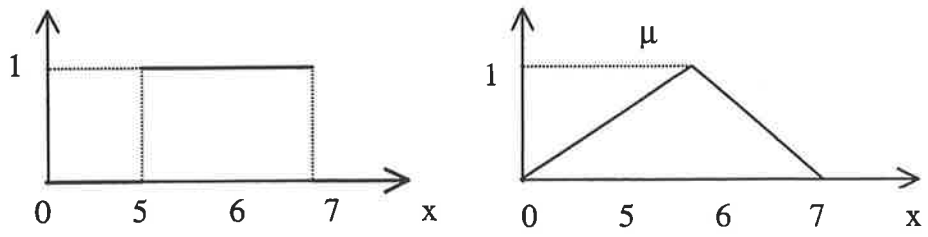


Figure 5-1: Membership functions for a crisp set and fuzzy set.

If X is a finite set $\{x_1, x_2, \ldots x_n\}$ a fuzzy set on X is expressed as

$$A = \frac{\mu_A(x_1)}{x_1} + \quad \ldots \quad + \frac{\mu_A(x_n)}{x_n}$$

$$= \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i} \tag{5.2}$$

when x is not finite

$$A = \int_x \frac{m_A(x)}{x} \tag{5.3}$$

5.3 Fuzzy Set Operations

- **Union and Intersection of Fuzzy sets**

$$\left.\begin{aligned} \forall x \in X,\ \mu_{A \cup B}(x) &= \max(\mu_A(x), \mu_B(x)) \\ \forall x \in X,\ \mu_{A \cap B}(x) &= \min(\mu_A(x), \mu_B(x)) \\ \forall x \in X,\ \mu_A(x) &= 1 - \mu_A(x) \end{aligned}\right\} \tag{5.4}$$

Union and intersection of fuzzy sets A and B, and complement of fuzzy set A are shown in Figure 5-2.

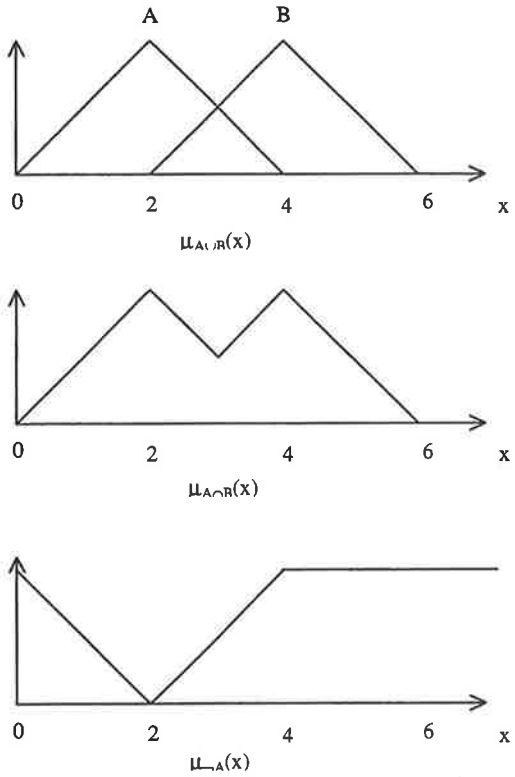


Figure 5-2: Union and intersection of fuzzy sets A and B, and complement of fuzzy set A.

Two fuzzy sets A and B are said to be equal (A=B)

$$\forall x \in X, \mu_A(x) = \mu_B(x) \tag{5.5}$$

- **Cardinality**

For a finite fuzzy set A the cardinality |A| is defined as

$$|A| = \sum_{x \in X} \mu_A(x) \tag{5.6}$$

- **The support of fuzzy set**

The support of fuzzy set A is the ordinary subset of X

$$\text{Supp } A = \{\forall x \in X, \mu_A(x) > \mu_B(x)\} \tag{5.7}$$

The elements of x such that $\mu_A(x) = 0.5$ are the crossover points of A.
The height of A is

$$\text{hgt}(A) = \sup_{x \in X} \mu_A(x) \tag{5.8}$$

That is, the least upper bound of $\mu_A(x)$. A is said to be normalized if and only if $\exists\, x \in X$

$$\mu_A(x) = 1 \tag{5.9}$$

$$\Rightarrow \quad hgt(A) = 1$$

$$\forall x \in X,\ \mu_{A\cup B}(x) = \max\left(\mu_A(x), \mu_B(x)\right)$$
$$\forall x \in X,\ \mu_{A\cap B}(x) = \min\left(\mu_A(x), \mu_B(x)\right)$$

where $\mu_{A\cup B}(x)$ and $\mu_{A\cap B}(x)$ are membership functions of A∪B and A∩B respectively.

- **Inclusion of fuzzy set**

A is said to be included in B($A \subseteq B$) if and only if

$$\forall\, x \in X,\ \mu_A(x) \le \mu_B(x) \tag{5.10}$$

- **α-level set**

The crisp set of elements that belongs to the fuzzy set A at least to the degree α is called α-level set.

$$A_\alpha = \{\, x \in X \mid \mu_A(x) \ge \alpha \,\} \tag{5.11}$$

$$A'_\alpha = \{\, x \in X \mid \mu_A(x) > \alpha \,\} \tag{5.12}$$

is called strong α-level set or α-cut.

- **Normal Fuzzy set**

A fuzzy set with a membership function that has a grade unity or 1 is called normal fuzzy set.

- **Convexity**

Convexity conditions are defined with reference to the membership function rather than the support of a fuzzy set. A fuzzy set is convex if

$$\mu_A\left[\lambda x_1 + (1-\lambda)x_2 \ge \min(\mu_A(x_1), \mu_A(x_2)\right], \quad x_1, x_2 \in X \quad \lambda \in [0,1] \tag{5.13}$$

Alternatively a fuzzy set is convex if all α-level sets are convex.

5.4 Properties of Fuzzy Sets

(1) The algebraic sum C = A + B is defined as

$$C = \{x, \mu_{A+B}(x) \mid x \in X\}$$
where
$$\mu_{A+B}(x) = \{\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)\} \tag{5.14}$$

(2) The bounded sum C = A⊕B is defined as

$$C = \{x, \mu_{A\oplus B}(x) \mid x \in X\}$$
where
$$\mu_{A\oplus B}(x) = \min\{1, \mu_A(x) + \mu_B(x)\} \tag{5.15}$$

(3) The bounded difference C = A-B is defined as

$$C = \{x, \mu_{A-B}(x) \mid x \in X\}$$
where
$$\mu_{A-B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\} \tag{5.16}$$

(4) The algebraic product of fuzzy sets C = A.B is defined as

$$C = \{x, \mu_A(x) \cdot \mu_B(x) \mid x \in X\} \tag{5.17}$$

5.5 Fuzzy Numbers

A convex and normalized fuzzy set for which each α-level set is a closed interval is called a fuzzy number. The *π-function* and the *S-function* are the two commonly known examples of fuzzy numbers. The π-function and S-function are shown in Figure 5-3 and Figure 5-4 respectively.

The π-function is defined as follows

$$\pi(x, \alpha, \beta) = \begin{cases} S(x; \beta - \alpha, \beta) & \text{if } x \prec \beta \\ 1 - S(x; \beta, \beta + \alpha) & \text{if } x \succ \beta \end{cases}$$

$$\pi(x;\ \alpha,\ \beta) = \frac{1}{1 + \left[x - \alpha/\beta\right]^2} \tag{5.18}$$

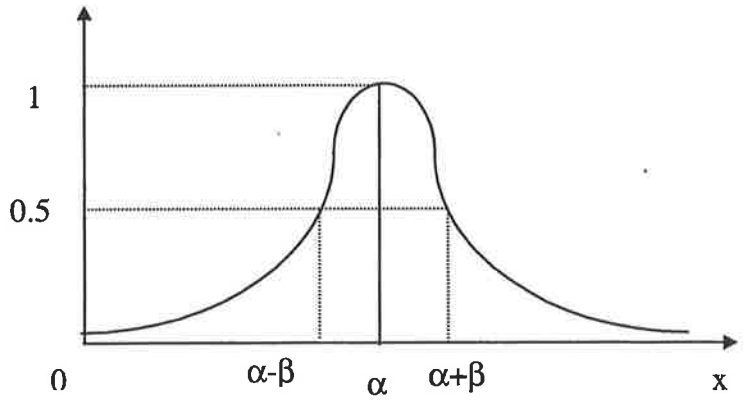


Figure 5-3: The π-function.

(b) The S-function is defined as follows

$$S(x,\alpha,\beta) = \begin{cases} 0 & \text{if } x \prec \alpha \\ 2\left(\dfrac{x-\alpha}{\beta-\alpha}\right)^2 & \text{if } \alpha \le x \prec \dfrac{\alpha+\beta}{2} \\ 1-2\left(\dfrac{x-\beta}{\beta-\alpha}\right)^2 & \text{if } \dfrac{\alpha+\beta}{2} \le x \prec \beta \\ 1 & \text{if } x \ge \beta \end{cases}$$

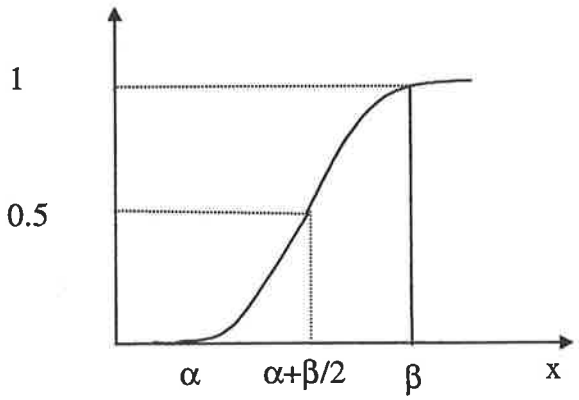


Figure 5-4: The S-function

5.6 The Extension Principle

One of the most important tools of fuzzy set theory introduced by Zadeh is the extension principle which allows the generalisation of crisp mathematical concepts to the fuzzy set and extends point-to-point mappings to mappings for fuzzy sets. Let X be a Cartesian product of universes, $X = X_1 \times X_2 \times \ldots \times X_r$ and $A_1, \ldots A_r$ be r fuzzy sets in $X_1 \times X_2 \times \ldots \times X_r$ respectively.

The Cartesian product of $A_1, \ldots A_r$ defined by

$$A_1, \ldots A_r = \int_{X_1 \times X_2 \times \ldots \times X_r} \frac{\min(\mu_{A_1}(x_1), \ldots, \mu_{A_r}(x_r))}{(x_1, \ldots x_r)} \tag{5.20}$$

Let f be a mapping from $X = X_1 \times X_2 \times \ldots \times X_r$ to a universe Y such that

$$Y = f(x_1, \ldots, x_r)$$

The extension principle allows us to induce from r fuzzy sets A_i a fuzzy set B on Y through f such that

$$\mu_B(y) = \sup_{x_1 \ldots, x_r} \min(\mu_{A_1}(x_1), \ldots, \mu_{A_r}(x_r)) \tag{5.21}$$

$$y = f(x_1 \ldots, x_r)$$

$\mu_B(y) = 0$ if $f^{-1}(y) = \phi$

where $f^{-1}(y) = \phi$ is the inverse image of y. $\mu_B(y)$ is the greatest among the membership values of the realizations of y using r-tuples. Zadeh usually writes the above equation as

$$B = f(A_1, \ldots A_r) = \int_{X_1 \times X_2 \times \ldots \times X_r} \frac{\min(\mu_{A_1}(x_1), \ldots, \mu_{A_r}(x_r))}{f(x_1, \ldots x_r)} \tag{5.22}$$

where sup operation is implicit.

5.7 The Resolution Principle

Let A be a fuzzy set in the universe of discourse U. Then the membership function of A can be expressed in terms of the characteristic functions of its α-cuts according to

$$\mu_A(x) = \sup_{\alpha \in (0,1]} [\alpha \wedge \mu_{A_\alpha}(x)] \qquad \forall x \in U,$$

where $\wedge$ denotes the min operation and $\mu_{A_\alpha}(x)$ is the characteristic function of the crisp set A_α.

$$\mu_{A_\alpha}(x) = \begin{cases} 1 & \text{if and only if } x \in A_\alpha \\ 0 & \text{otherwise} \end{cases}$$

Proof: Let $\vee$ denote the max operation. Since $\mu_A(x) \geq \alpha$ we have,

$$\begin{aligned} \sup_{\alpha\in(0,1]} [\alpha \wedge \mu_{A_\alpha}(x)] &= \sup_{\alpha\in(0,\mu_A(x)]} [\alpha \wedge \mu_{A_\alpha}(x)] \vee \sup_{\alpha\in(\mu_A(x),1]} [\alpha \wedge \mu_{A_\alpha}(x)] \\ &= \sup_{\alpha\in(0,\mu_A(x)]} [\alpha \wedge 1] \vee \sup_{\alpha\in(\mu_A(x),1]} [\alpha \wedge 0] \\ &= \sup_{\alpha\in(0,\mu_A(x)]} \alpha \\ &= \mu_A(x) \end{aligned}$$

The Resolution Principles states that the fuzzy set A can be expressed as in terms of its α-cuts without resorting to the membership function. Conversely, a fuzzy set A can also be decomposed into αA_α and it can be expressed as the union of these αA_α, $\alpha \in (0,1]$. The following relational equation so called the resolution principle. Figure 5-5 illustrates the concept of decomposition of fuzzy set.

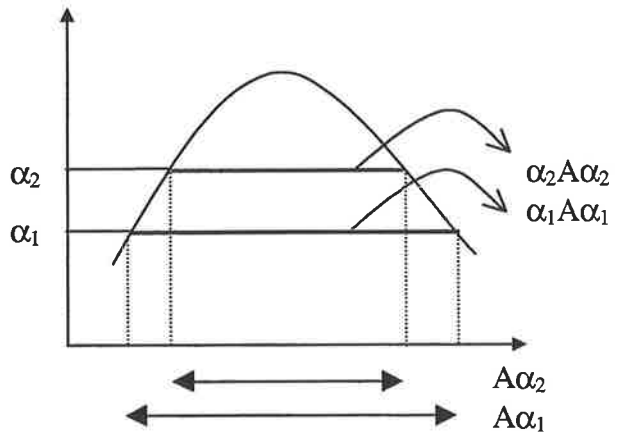


Figure 5-5: Decomposition of a fuzzy set.

5.8 Fuzzy Relational Equations

- **Composition of fuzzy relations**

Let Q be a fuzzy relation from X to Y and let R be a fuzzy relation from Y to Z we define $R \circ Q$, a fuzzy relation from X to Z by

$$\mu_{R\circ Q}(A,C) = \sup_{B\subseteq Y}\left[\mu_Q(A,B) \wedge \mu_R(B,C)\right] \tag{5.23}$$

when A is a fuzzy subset of X the sup-min composition of R with A yielding

$$B = R \circ A \tag{5.24}$$

which is a fuzzy subset of Y defined by

$$\begin{aligned} m_{R\circ A}(y) &= \sup_{x\in X}\left[\min(\mu_A(x),\mu_R(x,y)\right], \forall y \in Y \\ m_{R\circ A}(y) &= \bigvee_{x}\left[\mu_A(x) \vee \mu_R(x,y)\right], \quad \forall y \in Y \end{aligned} \tag{5.25}$$

5.9 Fuzzy Rule-Based Systems and Fuzzy Inference

A fuzzy system is characterized by a set of linguistic statements based on expert knowledge. The expert knowledge is usually in the form of "*if-then*" rules which are easily implemented by fuzzy conditional statements in fuzzy logic. In fuzzy logic and approximate reasoning, there are two important fuzzy implications inference rules named the *generalised modus ponens* (GMP) and the *generalised modus tollens* (GMT):

premise 1: x is A'
premise 2: if x is A then y is B,
consequence: y is B'

premise 1: y is B'
premise 2: if x is A then y is B,
consequence: x is A'

where A, A′, and B, B′ are fuzzy sets or relations also known as fuzzy predicates.
The fuzzy implication inference is based on the compositional rule of inference for approximate reasoning suggested by Zadeh.

Every rule has a weight, which is a number between 0, and 1 and this is applied to the number given by the antecedent. It involves 2 distinct parts. First evaluating the antecedent, which in turn involves fuzzifying the input and applying any necessary fuzzy operators and second applying that result to the consequent known as implication. If the premise is true, then the conclusion is true (for 2-valued or binary logic) In the case of fuzzy statement if the antecedent is true to some degree of membership then the consequent is also true to that same degree

A fuzzy inference is the powerful tool as a modeling method for complex and imprecise systems. The fuzzy inference is also a very suitable method to represent the knowledge of human experience. However, the fuzzy inference does not possess essentially the learning mechanisms or algorithms in it. We can acquire the knowledge for the objective systems in the form of fuzzy inference rules form the given data.

5.10 Aggregation of Fuzzy Rules

Most rule-based systems involve many rules. The process of obtaining the overall consequent from the individual consequent contributed by each rule in the rule-base is known as aggregation of rules. In determining an aggregation strategy, two simple extreme cases exit.

(1) Conjunctive system of rules: In the case of a conjunctive system of rules that must be jointly satisfied, the rules are connected by "and" connectives. In this case the aggregation output y is found by the fuzzy intersection of all individual rule consequent

$$y = y^1 \cap y^2 \cap y^3 \cap \ldots \cap y^r \tag{5.26}$$

which is defined by the membership function

$$\mu_y = \min(\mu_y{}^1(y), \mu_y{}^2(y), \mu_y{}^3(y), \ldots \mu_y{}^r(y)) \qquad \text{for } y \in Y \tag{5.27}$$

(2) Disjunctive system of rules: For the case of a disjunctive system of rules where the satisfaction of at least one rule is required, the rules are connected by the "or" connectives. In this case the aggregated output is found by the fuzzy union of all individual rule contributions, as

$$y = y^1 \cup y^2 \cup y^3 \cup \ldots \cup y^r \tag{5.28}$$

which is defined by the membership function

$$\mu_y = \max(\mu_y{}^1(y), \mu_y{}^2(y), \mu_y{}^3(y), \ldots \mu_y{}^r(y)) \qquad \text{for } y \in Y \tag{5.29}$$

5.11 Graphical Techniques of Inference

The following graphical procedure (Figure 5-6) illustrates analysis of two rules and can be easily extended and will hold for fuzzy rule-bases with any number of antecedents and consequent. Here the symbols A_{11} and A_{12} refer to the first and second fuzzy antecedents of the first rule, respectively, and the symbol B_1 refers to the fuzzy consequent of the second rule; the symbols A_{21} and A_{22} refer to the first and second fuzzy antecedents of the first rule respectively, and the symbol B_2 refers to the fuzzy consequent of the second rule; The minimum membership value for the antecedent propagates through to the consequent and truncates the membership function for the consequent of the rule. This graphical inference is done for each rule. Then the truncated membership functions for each rule are aggregated using the graphical equivalent of equation (5.27) for conjunction rules or equation for (5.29) disjunctive rules. In the following figure rules are disjunctive, so the aggregation operation max results in an aggregated membership function comprised of outer envelop of the individual truncated membership forms from each rule.

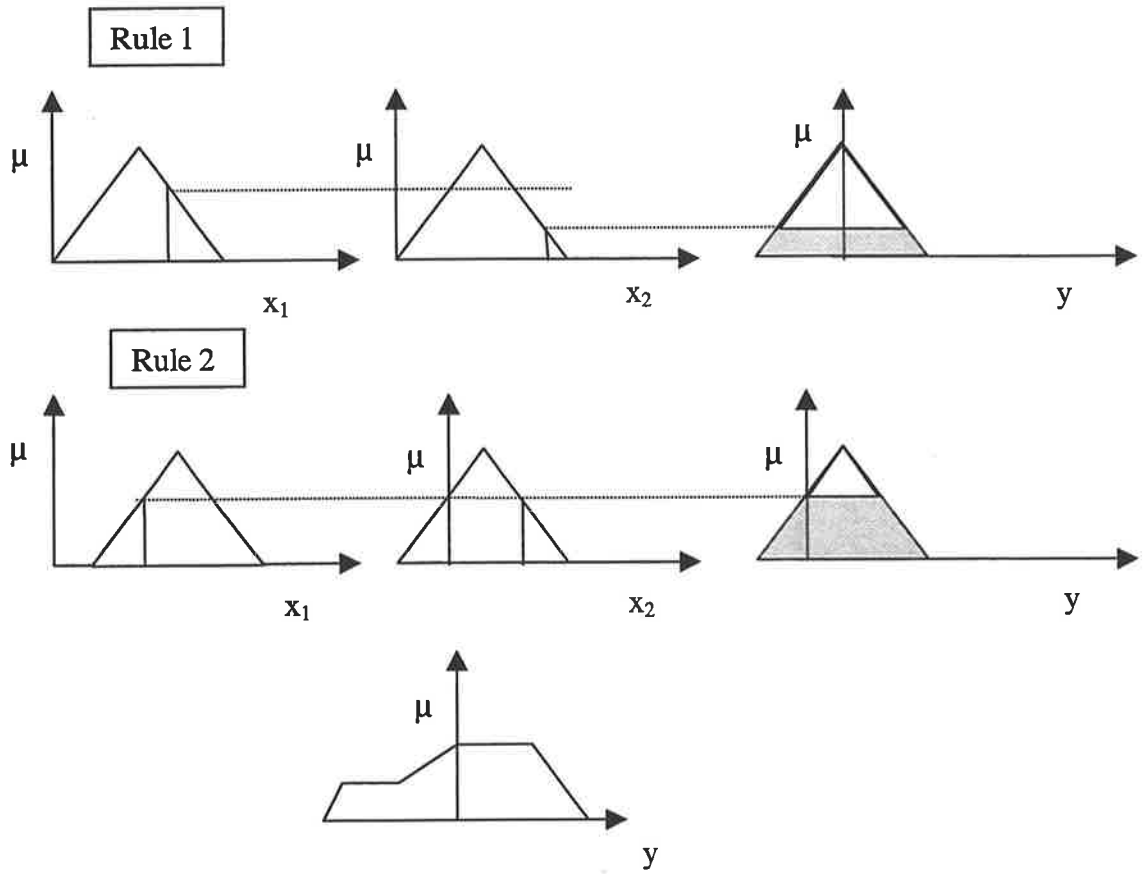


Figure 5-6: Graphical mamdani inference method.

- **Advantages of Fuzzy set theory**

1. One of the advantages of a fuzzy system based on such a fuzzy rule its comprehensibility. That is human users can easily understand each fuzzy if-then rule because its antecedent and consequent parts are represented by linguistic values. Fuzzy rules are also called linguistic rules, because they represent the way in which people usually formulate their knowledge about a given process. Fuzzy set techniques has low information and time complexity.
2. Fuzzy set techniques seem to be good solution for some of the problems that arise due to lexical imprecision.
3. Fuzzy set techniques are quite flexible

- **Disadvantages of Fuzzy set theory**

1. It is not always clear how to construct reasonable membership functions.
2. The choice of appropriate definitions for the operators can be problematic.

5.12 Conclusion

In this chapter, the basic concepts and notation of fuzzy sets are presented. A hybrid learning methodology that integrates fuzzy systems and genetic algorithms presented in chapter 9.

The basic aims of fuzzy logic is to provide a computational framework for knowledge representation and inference in an environment of uncertainty and imprecision. In such environments, fuzzy logic is effective when the solutions need not be precise and/or it is acceptable for a conclusion to have a dispositional rather than categorical validity. The importance of fuzzy logic derives from the fact that there are many real-world applications, which fit these conditions, especially in the realm of knowledge-based systems for the decision-making. In short, conventional methods are good for simpler problems, while fuzzy systems are suitable for complex problems or applications that involve human descriptive or intuitive thinking.

Chapter 6

Experimental Evaluation and Comparison

6.1 Introduction

An important application area of pattern recognition is medical diagnosis. Statistical pattern classification is particularly useful in cases where data measurements of disparate types are present, and there is no theoretical guidance on how to combine the quantities.

The advent of integrated information management system in health care is paving the way for efficient implementation and fielding of systems for computer-based medical decision support. Decision making under uncertainty is often fraught with great difficulty when the data on which the decision is based are imprecise and poorly linked to predict outcome. Clinical diagnosis is an example of such a setting because multiple, often unrelated, disease states can present with similar or identical historical, symptomalogic, and clinical data.

Heart disease is common cause of death in human and difficult to diagnose accurately. In this chapter, the use of multilayer perceptron trained with backpropagation was applied to the diagnosis of coronary artery disease disease. Other neural network classifiers were also tested on the data such as Modular network (MN), Radial basis function (RBF), Reinforcement learning (RL). Several variations of these classifers were tested on the same data. An attempt is made to formulate the neural network training criteria.

Currently, two most popular methods are those based on Bayes' theorem, where the relative likelihood of different diagnoses is computed from individual items of patient information; and expert system, which seek to mimic a clinicians reasoning process by encapsulating the knowledge that clinicians apply in a set of rules. A considerable number of methodologies such as discriminant analysis, logistic regression, recursive partition analysis and so forth have been developed in attempts to improve on the diagnostic accuracy of physicians in identifying the presence or absence of the disease. Previous approaches to diagnostics problems have been based on tree structured rule-based, statistical probability calculations, linear pattern matching. These clinical decision-making methods are based on a highly structured set of rules or statistical probability prediction that are dependent on the accuracy of input data. They suffered from slow response time and failure under missing information. Artificial neural networks perform non-linear statistical analysis and they can tolerate a considerable amount of imprecise and incomplete input data.

Neural networks are a good approach for solving diagnostic problems given the property that small variations in the input pattern such as inaccuracy, noise or any other reason do not result in misclassification.

6.2 ANNs applied to Coronary artery disease

The original database contained 920 records, each consisting of 76 attributes. All published experiments, however, use only 14 of these attributes. One of the attributes associated with each patient, an integer $x \in [0...4]$, counts the number of major vessels whose diameter has reduced by more than 50%. For classification, the objective is to use the 13 independent attributes to predict the absence ($x=0$) or presence ($1 \leq x \leq 4$) of the disease.

- **The data set**

This data set has the following interesting characteristics:

- Some attributes are missing in some records:
- 67.5% of records have at least one missing value, and 14.7% of the input attribute values are absent.
- It comes from four independent medical institutions. The database is often known as the Cleveland data set, as this portion of the data has no missing attributes and consequently has often been used on its own by the machine learning community.
- The records contain a mixture of real-valued, Boolean and enumerated values. A description of each variable is shown in Table 6.1.

Table 6-1: Description of the 13 predictor variables from the coronary artery disease data set; variable types are (R)eal, (E)numerated, (B)oolean.

Var.	Name	Type	Range
1	age	R	28-77
2	sex	B	0-1
3	chest pain type	E	typical angina, atypical angina, non-anginal pain, asymptomatic
4	resting blood pressure	R	80-200
5	cholestoral	R	0-603
6	fasting blood sugar > 120 mg/dl	B	0-1
7	resting electrocardiographic results	E	normal, abnormality, hypertrophy
8	maximum heart rate achieved	R	60-202
9	exercise induce angina	B	0-1
10	ST depression induced by exercise relative to rest	R	-2.6...6.2
11	slope of the peak exercise ST segment	E	upsloping, flat, downsloping
12	ca	R	0-3
13	thal	E	normal, fixed defect, reversible defect

6.3 Knowledge Representation by Neural Networks for Diagnostics

There were two main types of MLP networks: networks designed to classify all five possible diagnostic classes and networks designed to recognize a single diagnostic class. The non-linear multilayer perceptron trained with backpropagation was applied with 13 inputs, 1 layer of 50 hidden units and 1 output neuron. The input clinical variables are 8 symbolic and 6 numeric. The training parameters of learning rate and momentum were set at 0.1 and 0.9 respectively. The network was trained by dividing the available data into a training set and a test set. The output produced by the neural network was then compared to the documented output.

Thus in order to use the given training set with back-propagation it is essential for it to be pre-processed so that the components of input and output vectors are represented as activation levels. Some of the attributes are non-numeric (symbolic) and some are numeric. These attributes can be easily normalised between 0 and 1. To deal with symbolic attributes such as type of chest pain, it is necessary to map each value of each attribute into a unique integer, beginning with zero and working upwards. A better way is to map these values of the symbolic attributes into sparce binary vectors, i.e. binary vectors that have only one bit set. Applied to the third attribute (chest pain type) from the training set (which has 4 different values) this would give us the mapping as follows:

typical angina	atypical angina	non - anginal pain	asymptomatic
↓	↓	↓	↓
1	0	0	0
0	**1**	0	0
0	0	**1**	0
0	0	0	**1**

Once we have mapped specific values into such binary vectors, we can turn an input into a satisfactory activation vector by simply normalising the numeric attribute values and replacing the symbolic attribute values with their given binary sequence. Having derived usable training set, we should decide what architecture we are going to use. The number of input units and output units is fixed by the form of the input and output vectors in the derived training set. The input vectors contain 13 components while the output vectors contains just one. Thus the network must have 13 input units and one output unit.

The following is a brief summary of the attributes used in the database.
#1 is the age in years;
#2 is the patient's sex:
value 0: female;
value 1: male;
#3 is the chest pain type:
value 1: typical angina;
value 2: atypical angina;

value 3: non-anginal pain;
value 4: asymptomatic;
#4 is the resting blood pressure (in mm Hg on admission to the hospital);
#5 is the serum cholesterol in mg/dl;
#6 give an indication for fasting blood sugar:
value 0: ≤ 120 mg/dl;
value 1: > 120 mg/dl;
#7 give a classification for the resting electrocardiographic results:
value 0: normal;
value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV);
value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria;
#8 is the maximum heart rate achieved during exercising;
#9 indicates whether exercise induced angina:
value 0: no;
value 1: yes;
#10 is the ST depression induced by exercise relative to rest;
#11 is the type of slope of the peak exercise ST segment:
value 1: upsloping;
value 2: flat;
value 3: downsloping;
#12 is the number of major vessels (0-3) colored by fluoroscopy;
#13 thal; not further explained:
value 3: normal;
value 6: fixed defect;
value 7: reversible defect;
#14 is the predicted attribute: diagnosis of heart disease (angiographic disease status):
value 0: no heart disease;
values 1, 2, 3, and 4: heart disease.

In a few instances an attribute value was missing. They are distinguished with the value -1.

The Back-propagation algorithm was used to a train feed forward MLP type neural networks. After presenting one training example to the 13 network inputs (this is the feedforward part), the error at the output is calculated by subtracting the desired output from the actual network output and taking the square of the difference. The errors for all examples in the training set are then added together and their sum is back propagated through the network, adjusting the weights of the connections between the processing elements of the network. Using a gradient descent learning algorithm the network error should decrease towards a minimum. The processing elements pass the sum of their inputs through a sigmoid function, limiting the neuron output between zero and one.

Several settings were used for the training set, for the learning period, and for the various network architectures in this study. To see how much influence the amount of processing elements in the hidden layer would have, experiments with both 20 and 50 units in the hidden layer were conducted.

Also the number of output units was varied; networks with just one, with two, and with five outputs were used. The networks with five outputs were trained with data containing five target categories: one category for "no heart disease" and four others for

different grades of "heart disease presence". The networks with two outputs were trained with data containing only two categories: "absence" versus "presence". And the networks with just one output were trained in two ways: using five categories and two categories. In the first case the five categories {0,1,2,3,4} were scaled to values between zero and one {0.00, 0.25, 0.50, 0.75, 1.00}, as the network output is limited between those two values. In the second case the network output should be low for one category ("absence") and high for the other ("presence").

The networks were trained with three different training sets. First, the full database of 303 examples was used as training set for the learning of the networks, and the performance of the networks was tested with the same set, as there were no other data to measure the performance with. Second, the original database was split into a training set part with 253 examples used for training, and a testing set part with 50 instances used for measuring the network performance. The reason for this is that one cannot test the ability of a neural network to generalise if one uses the same set of data for both training and testing. If a network performs as well on a different testing set as on its training set, then the network learned to generalise well. If however a network performs much worse on a testing set in comparison with its performance on the data set with which it was trained, then the network did not learn to generalise well enough. This happens for example if the number of hidden neurons is too large and the learning period was too long. Third, a training set of 202 examples for the learning of the networks, and a testing set of 101 examples to measure their performances were used. Both sets together form the full original data set again because a testing set that contains only 50 examples is quite small. Because it is not always very clear when the learning of a network should be stopped, three learning periods of 5000 epochs each per network were used. After each learning period, the network was tested.

Even if the output error during learning of a network is still decreasing, this does not necessarily mean that the performance of that network is still improving. The network output error might be decreasing, but the number of misclassifications could be increasing. This depends on the way the error is calculated and on the definition of misclassification that is used in that particular case. Further, the number of misclassifications might be decreasing when testing with the same data set as which the network was trained with, whilst at the same time the number of misclassifications could be increasing when using a different data set for testing the network performance.

- **Output Representation**

A total of 72 experiments were performed. Each network was used to perform three experiments: The number of neural networks used to make a diagnosis on possible heart disease patients is 24. Each network in three experiments per network were examined: first after 5,000 epochs of learning, then after 10,000 epochs, and finally after a total of 15,000 epochs. Twelve networks had 20 processing elements in their single hidden layers, and the other twelve had 50 hidden neurons each. Six networks had a single output each and were trained for classifying in five classes: {0.00, 0.25, 0.50, 0.75, and 1.00}. Six other networks also had a single input, but they were trained for classifying in only two classes: {0.0, 1.0}. Six networks had two output units each, and they had to learn to classify in two classes also: {10, 01}. Finally, there were six networks with five outputs each, learning to classify in five classes again: {10000, 01000, 00100, 00010, and 00001}. For every type of network (different number of hidden units, different number of output units, different number of classes to learn) three neural networks,

were trained with different sizes of training sets and testing sets: 303+0, 253+50, and 202+101

The output class is either healthy (class 0) or with heart disease (class 1). The target values for the output were coded as 0 for confirmed normal and 1 for confirmed abnormal. The testing of the network was achieved by using the weights derived in the training set and applying the new pattern to the network to which it has not been exposed. The network was tested on 50 patients. All network training was done in a supervised fashion, which means that the inputs and desired outputs were known during the training process. Patients were selected from the Cleveland Clinic Foundation.

6.4 Simulation Results

The accuracy of each network was found by counting the number of misclassifications, using only two categories: "no heart disease present" and "heart disease present". The number of misclassifications divided by the total number of instances in the testing data set, subtracted from 100 per cent, gives the actual accuracy.

Misclassifications appear in two ways: a *false alarm* occurs when the neural network detects a heart disease whilst the patient in question does not have a heart disease; a *missed detection* occurs when the network fails to detect the patient's heart disease. The full results of the experiments are presented at the end of this chapter.

The best performing network from these experiments was a network with 50 processing elements in its hidden layer, and with a single output that learned with two classes and 202 examples. After 5,000 epochs of learning a testing set of 101 examples was tested, and it showed an accuracy of 87.1 %. The following Table 6-2 summarises these results.

Table 6-2: Best results for different network properties and learn/test methods

	Tested with →	**Train set (303 ex.)**		**Test set (50 ex.)**		**Test set (101 ex.)**	
1	Property ↓	epochs	score	epochs	score	epochs	score
2	20 hidden units	15,000	85.8	5,000	84.0	5,000	84.2
3	50 hidden units	10,000	85.8	5,000	84.0	4,000	87.1
4	1 output, 5 classes	15,000	84.5	5,000	84.0	5,000	84.2
5	1 output, 2 classes	10,000	85.8	5,000	84.0	4,000	87.1
6	2 outputs, 2 classes	10,000	84.8	5,000	82.0	10,000	83.2
7	5 outputs, 5 classes	10,000	84.5	5,000	82.0	15,000	83.2

From this table it can be seen that the performance increased after 5,000 learning epochs, if one would only look at the results on the training set. Looking at the results if the networks are tested with a test set that is different from the training set, shows that the real performance almost never increased after 5,000 learning epochs. Further we see a minor indication that a larger number of hidden neurons causes slightly better results.

If we look at the number of outputs and classes, notice that the best number of outputs seems to be just one, and that learning for two classes gives better results than learning for five classes.

It is obvious from this study that the present practice of choosing manually the architecture of the neural network is not only time consuming but also it does not give

the optimum performance. Evolutionary computing techniques can provide a solution to this problem.

6.5 The Stopping Criterion

In order to prevent overtraining, when to stop training the network is examined. If the network is overtrained, it loses its ability to generalise; this means that the network performance is significantly worse when new data is fed to the network, (data that was not used for training), instead of the data in the training set.
A solution to prevent overtraining is so-called cross-validation. This means that during training the network error is not only calculated for the training data, but also for an independent test data set, which has the same statistical properties as the train data set. After every n epochs the network parameters are written to a file, so that afterwards we can retrieve the network that had a minimal error on the test data. In Figure 6-1 and 6-2 two examples can be seen. These figures shows the error graphs of two different networks.

Both the error graph for the training data and the one for the testing data are shown. One can easily see that, although the error is still decreasing for the training data, the error for the testing data slowly increases after it reached its minimum. The network configuration with the best generalising behaviour is the one that was saved when the test error was minimal.

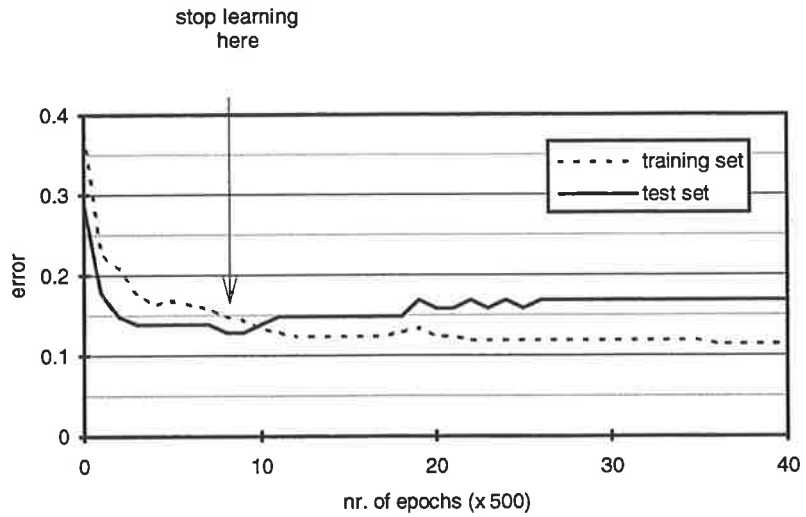


Figure 6-1: When to stop learning (13-50-1 network, training set 202 examples, test set 101 examples)

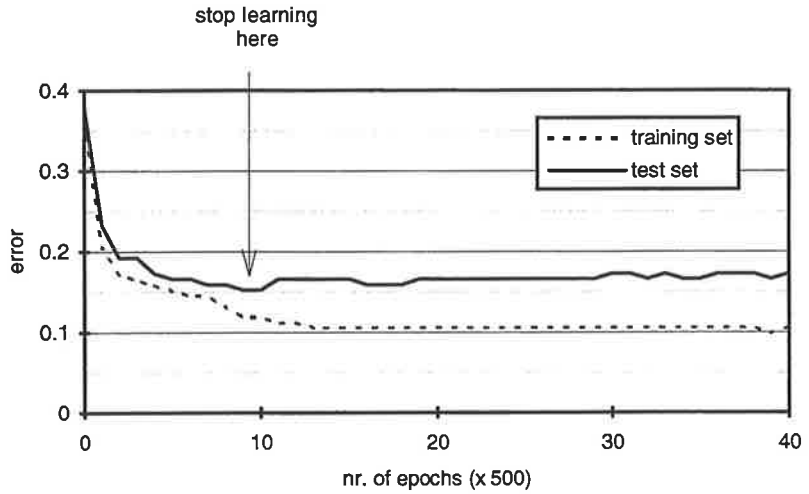


Figure 6-2: When to stop learning (13-50-1 network, training set 202 examples, test set 101 examples)

6.6 Performance of other Statistical Methods reported in the literature

Table 6-3 shows performance of other statistical methods reported in the literature. Comparison of percent correct classification of recognition on the test set using various Neural networks are shown in Table 6-4.

Table 6-3: Performance of other statistical methods reported in the literature.

	Methods	% Correct Classification
1	C4	74.8
2	NT Growth	77
3	CLASSIT Clustering	78.9

Best of the neural networks from experiments: 87.1% accuracy.

Table 6-4: Comparison of percent correct classification of recognition on the test set using various Neural networks.

	Network	Learning Rule	Transfer function	# of Input PEs	# of Hidden PEs	# of Outputs PEs	% correct classification
1	MLP	Delta	Sigmoid	13	50	1	87.1
2	MN	Delta	Sigmoid	13	5	1	82.82
3	RBF	NormCum	Gaussian	13	5	1	75
4	RL	DRS	Sigmoid	13	5	1	82.81

6.7 The Dual Spiral Problem

In this problem, the points of the 2 classes are on spirals in each other. We have experimented with 1 and 1.5 loops of the spiral round the center. One class is called "0" and the other is called "1". On each loop,each class has equidistant 100 points, so 200 examples with 1 loop and 300 with 1.5 loops. All the networks start with the same random weights. Globel learning were used.

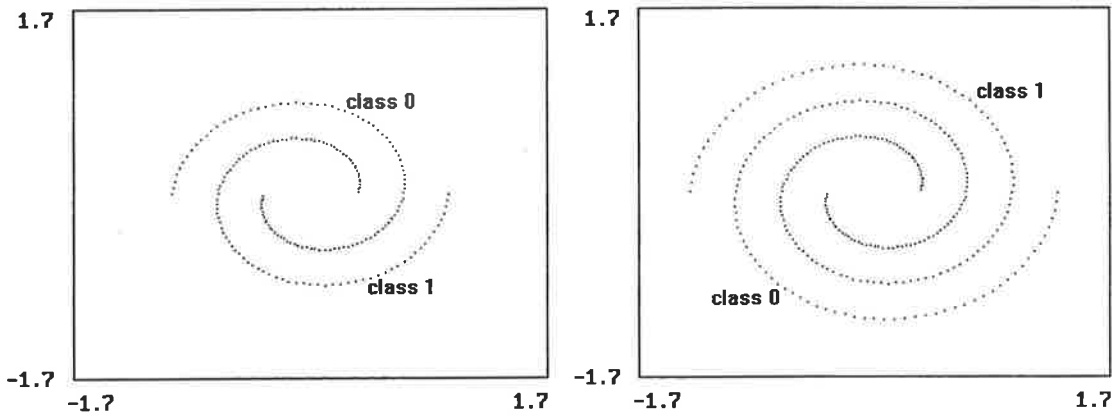


Figure 6-3: The dual spiral problem

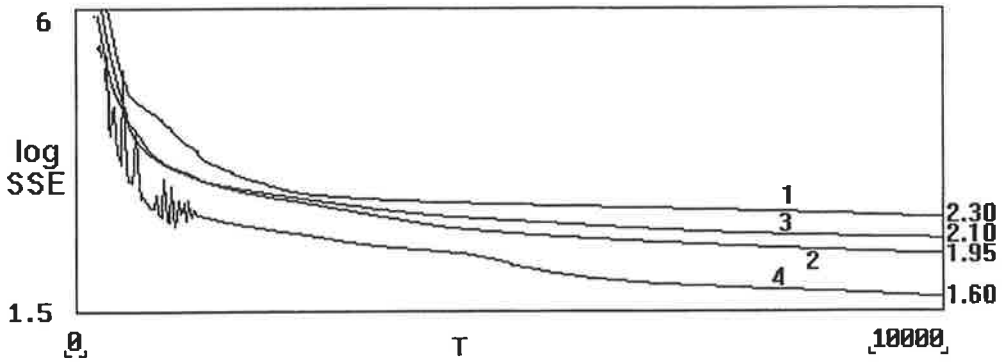


Figure 6-4: log of sum squared error with 4 different combinations of learning rate and momentum for 1 13-5-4-1 network.

1 = (LR:0.01,Mom = 0.9),
2 = (LR:0.02 , Mom = 0.8),
3 = (LR:0.02 , Mom = 0.8),
4 = (LR:0.1 , Mom = 0.6),
5 =(LR:0.02 , Mom = 0.8),

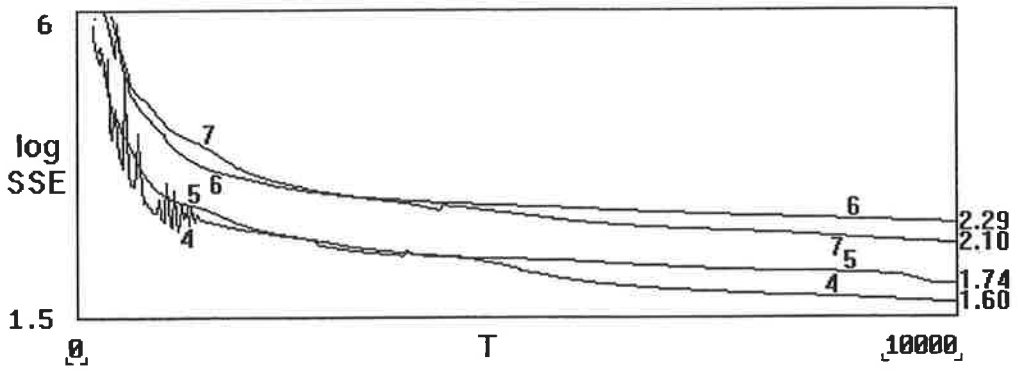


Figure 6-5: log of sum squared error with 4 different combinations of learning rate and momentum for 1 13-5-4-1 network.

4 = (LR:0.1,Mom = 0.6),
5= (LR:0.1 , Mom = 0.5),
6 = (LR:0.1 , Mom = 0.3),
7 = (LR:0.1 , Mom = 0.1),

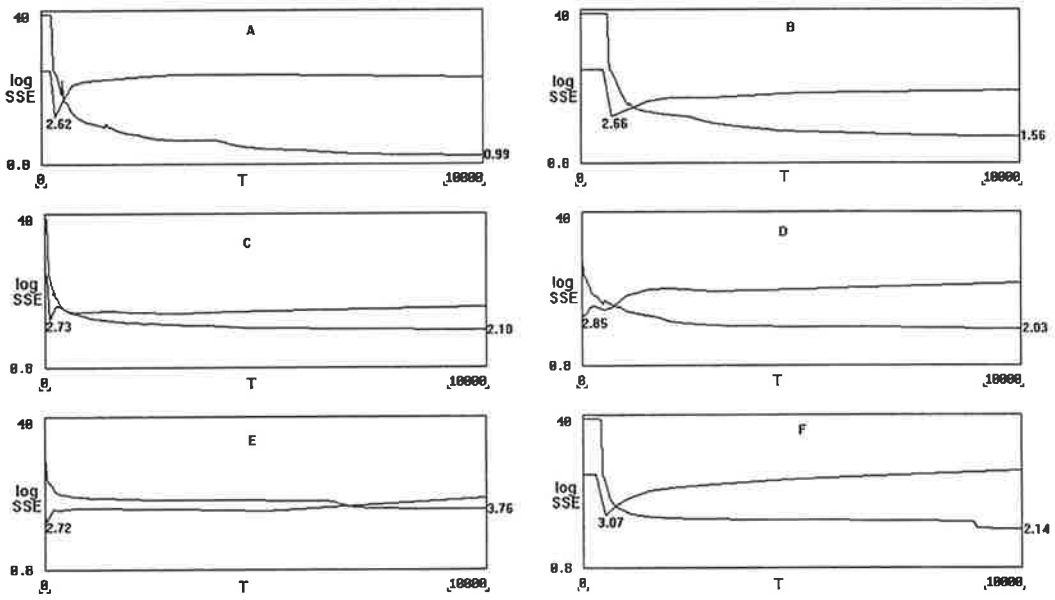


Figure 6-6: log SSE curves of train and test set with LR = 0.1 and Mom=0.6 (except for B and D, Mom=0.5) for the following networks: A 13-7-7-1, B: 13-5-7-1,C=13-5-4-1, D: 13-5-2-1, E=13-3-5-1,F=13-5-1

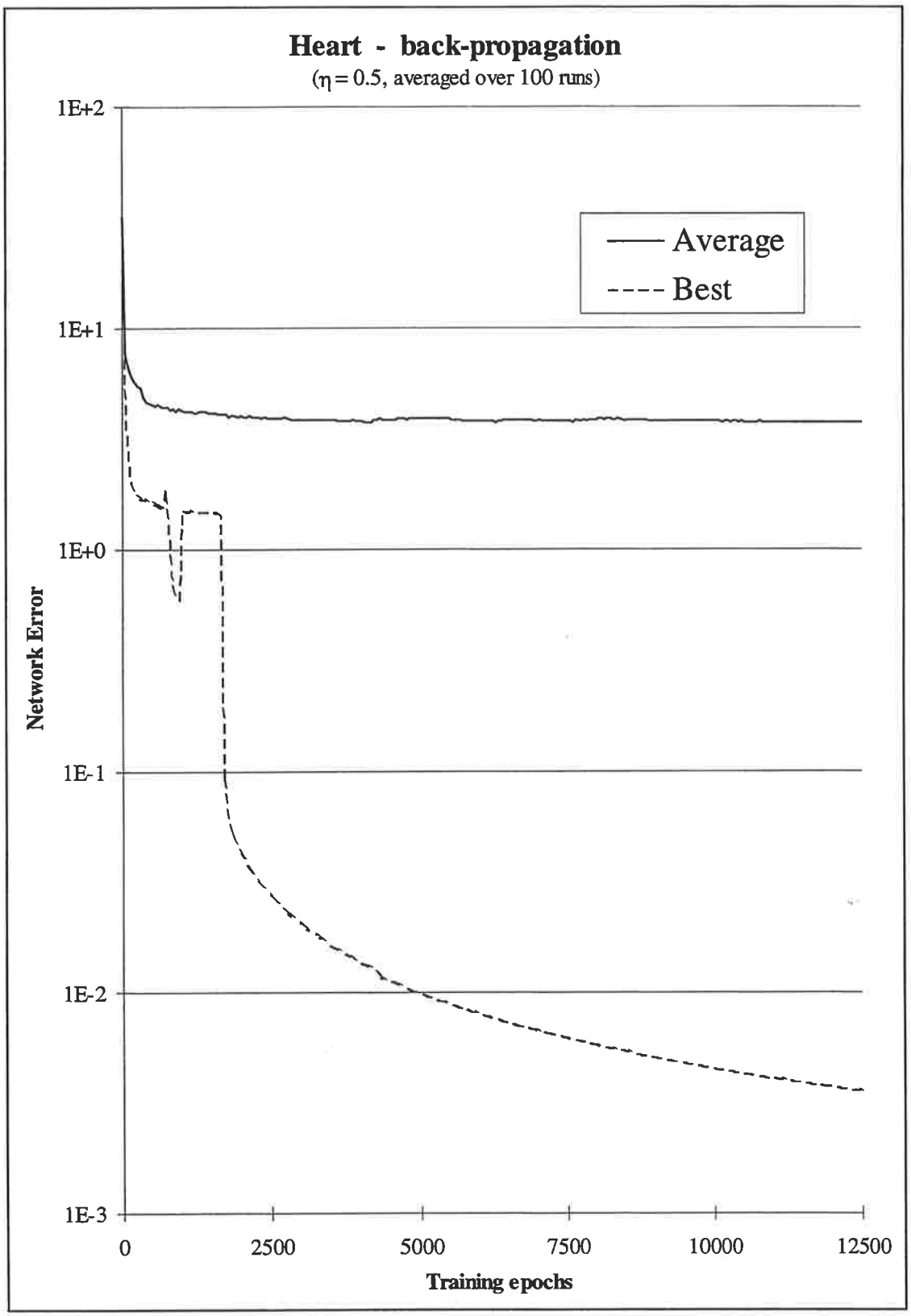


Figure 6-7: Mean square error graph

	Network structure			Training & testing parameters				% accuracy after # epochs	
1	# inputs	#hidden	#output	#classes	train set	test set	5,000	10,000	15,000
2	13	50	1	5	303	303	81.5	84.2	84.5
3	13	50	1	5	253	253 50	82.6 84.0	85.8 84.0	86.6 82.0
4	13	50	1	5	202	202 101	85.1 84.2	87.1 84.2	87.1 84.2
5	13	20	1	5	303	303	82.8	83.5	84.2
6	13	20	1	5	253	253 50	82.6 82.0	85.8 82.0	86.2 82.0
7	13	20	1	5	202	202 101	85.6 80.2	87.1 83.2	87.6 84.2
8	13	50	1	2	303	303	83.2	85.8	85.8
9	13	50	1	2	253	253 50	83.8 84.0	87.4 84.0	87.0 84.0
10	13	50	1	2	202	202 101	86.1 85.2	88.1 83.2	88.6 83.2
11	13	20	1	2	303	303	83.5	84.8	85.8
12	13	20	1	2	253	253 50	84.6 84.0	86.6 82.0	87.0 84.0
13	13	20	1	2	202	202 101	85.2 84.2	87.6 84.2	88.6 83.2
14	13	50	2	2	303	303	82.8	84.8	84.2
15	13	50	2	2	202	202 101	83.7 80.2	85.6 81.2	86.1 80.2
16	13	20	2	2	303	303	83.5	84.2	84.8
17	13	20	2	2	253	253 50	83.8 82.0	87.0 78.0	85.4 80.0
18	13	20	2	2	202	202 101	83.7 81.2	85.1 83.2	86.1 82.2
19	13	50	5	5	303	303	81.2	84.5	83.8
20	13	50	5	5	253	253 50	83.4 82.0	85.4 80.0	85.0 80.0
21	13	50	5	5	202	202 101	82.7 81.2	87.1 80.2	86.1 83.2
22	13	20	5	5	303	303	82.8	84.2	83.8
23	13	20	5	5	253	253 50	83.0 78.0	85.0 76.0	85.0 80.0
24	13	20	5	5	202	202 101	85.1 75.2	85.6 80.2	85.6 82.2
25	13	400	1	2	202	202 101	90.1 82.2	(after 50,000	 epochs)

6.8 Conclusion

In this chapter the use of artificial neural networks in a real-world diagnosis problem of coronary artery disease was examined. Performance comparison with various neural networks such as modular networks, radial basis function, reinforcement learning have been made. Also, an attempt is made to formulate the neural network training criteria in medical diagnosis. It is a usual practice to stop the training of a neural network as soon as the training of the neural network error reaches to a specified value. It is shown that the present approach is not reasonable and does not give accurate results. The approach presented in this chapter can save valuable training time.

Different input and output representations are derived and performance under each is analysed. From the results of experiments, the following conclusions can be drawn. First, we can see that the ability to classify correctly decreases slowly with the increase in the number of output neurons. Secondly, the number of classes for which the network is trained is best set to two, if the performance after learning is determined concerning only two categories as well. Thirdly, the numbers of hidden neurons in the simulations do not seem to make much difference, regarding the results, although the experiments with 50 hidden neurons instead of 20 gave slightly better results. Finally, if we compare at the accuracy of the simulated neural networks with performance of other statistical methods reported in the literature we see quite a large difference in favour of the neural networks.

Chapter 7

Experimental Results with Hybrid System

7.1 Introduction

This chapter presents a hybrid learning methodology that integrates genetic algorithms (GAs) and the back-propagation algorithms (BP), which should always find the correct global minima without getting stuck at local minima. In the hybrid learning algorithm back-propagation can be used as the training method with genetic algorithms used to escape from local optima. Thus making use of both genetic algorithms and back-propagation.

The back-propagation algorithm is a well-known method of training MLP networks. It involves a search in the weight space for the optimum weights, which minimise the error between the target output and the actual output of the network. This method is simple and easy to implement. However, it is dependent on several parameters, such as the learning rate and a momentum term. It is also sensitive to local minima. Since it is gradient descent method it has a tendency to get stuck in local minima of the error surface. None of these single methods has an answer for all the optimisation problems.

The genetic algorithm is incorporated in the back-propagation algorithm to find the best set of weights for mapping the input data to the output data. The performance of genetic algorithms does not depend on parameters such as the learning rate or momentum. Genetic algorithm is well suited to optimise functions having a large number of variables. Due to their ability to search the entire solution space simultaneously, GAs offer a low possibility of getting stuck at local minima.

One of the drawbacks of genetic algorithms is that as they get closer to the solution, the speed of convergence decreases. The genetic/back-propagation algorithm optimises the network's weights using a genetic algorithm. However, to overcome the slow convergence of the genetic algorithm in the final stage of the optimisation process, a switch can be made to the back-propagation algorithm as soon as the genetic algorithm has located a near optimal weight configuration. The final convergence to the optimal solution can be performed using the back-propagation algorithm. This way one can exploit the strong points of both the genetic algorithm (GA), and the back-propagation algorithm (BP). All experiments were performed on a real world-data set of coronary artery disease. A detailed description of data set is already given in Chapter 6.

Various versions of the this method are presented and experimental results show that GA-BP algorithms are as fast as the back-propagation algorithm and do not get stuck in local minima. It involves the optimisation of the connection weights of the MLP architecture for solving a specified mapping of an input data set to the output data set. The weight optimisation was achieved using the well-known gradient-descent method and a genetic algorithm.

7.2 Brief Description of GA software

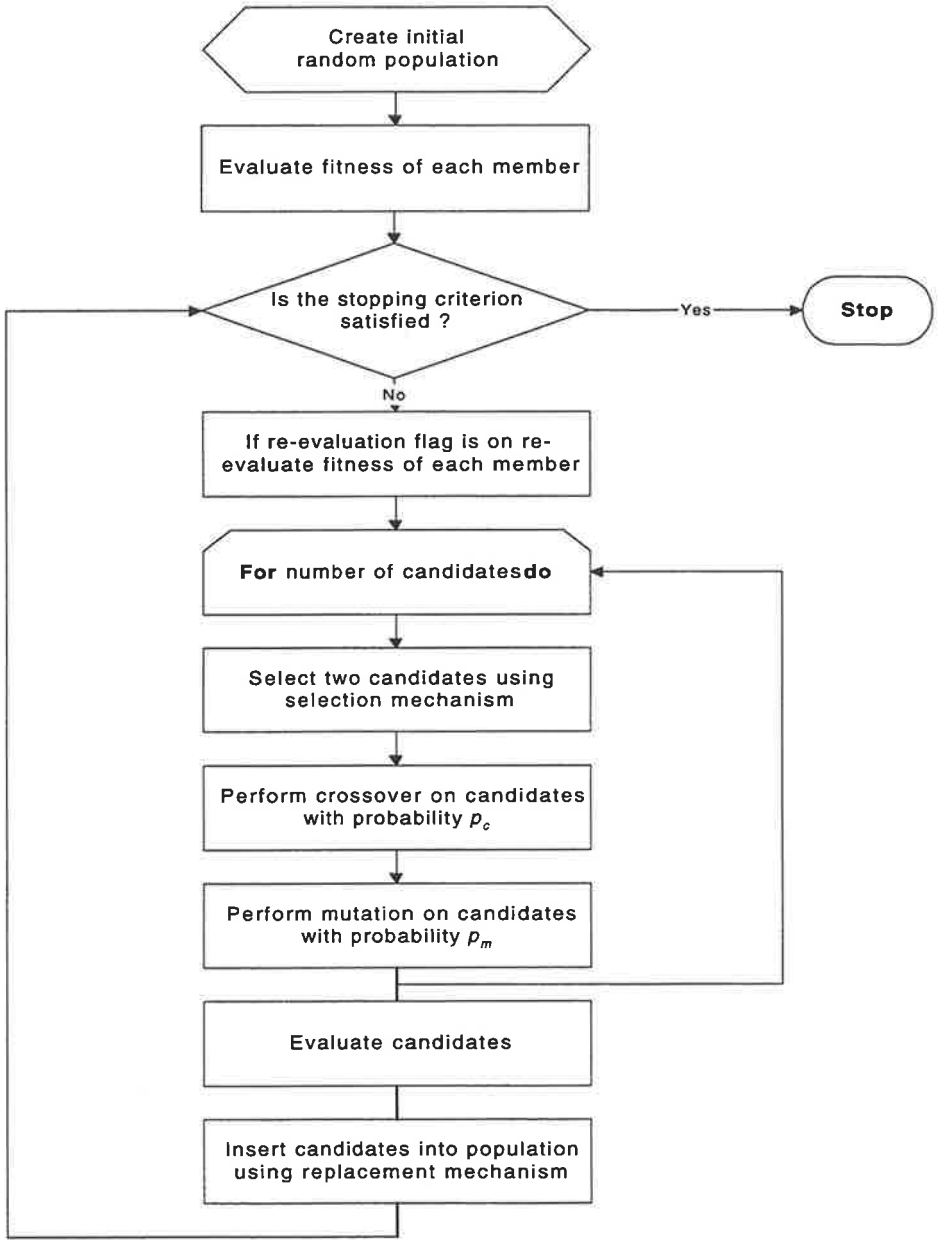


Figure 7-1: Flowchart for the GA software SUGAL

The GA software used was a genetic algorithm C-library called 'SUGAL' (v1.0), developed by A. Hunter at the University of Sunderland, England, and it was running on IBM 686. The basic working of SUGAL is illustrated in the flowchart of Figure 7-1.

In the course of a generation firstly pairs of individuals that serve as candidates to be included in the next generation are chosen using the selection mechanism. In the standard GA the number of candidates equals the population size; the candidates replace i.e. the complete population. An exception to this is when elitism is used. With elitism the number of candidates is equal to the population size minus one. As usual crossover is performed on the pair of candidates with probability p_C. With probability p_m, mutation is performed right after this. The candidates are then evaluated and they are

inserted back into the population using the replacement mechanism. In the standard GA the replacement mechanism is such that the candidates always replace the individuals in the population. This is known as unconditional replacement. SUGAL offers extra replacement strategies identical to the ones used in a steady-state genetic algorithm (SSGA); i.e. conditional/unconditional and ranked/unranked replacement. The standard GA can be transformed into a SSGA by decreasing the number of candidates to only one (or two).

The SUGAL software was changed regarding the implementation of the crossover operator. The standard procedure was chosen: a single chromosome is subject to mutation with probability p_m. This probabilistic implementation of the mutation operator where every chromosome has to undergo a 'test' to determine whether or not it should be mutated makes the program quite slow. A sequential version of the mutation operator done at the end of a generation where the number of chromosomes to be mutated is equal to integer value of the expected number of mutations would be much faster. Various replacement mechanisms however require that the chromosomes are evaluated before the replacement takes place (and evaluation, of course, must take place after the mutations are made).

A second change was made concerning the selection of the pair of candidates. In SUGAL it was possible for a single individual to be chosen both as the father and as the mother. In such a case the offspring are simply exact copies of the parent no matter what kind of crossover takes place. This has the effect of lowering the effective crossover rate and in populations with one superfit individual it could easily lead to premature convergence. The code was changed so that the father and mother chromosome could not be one and the same.

A special option in SUGAL is re-evaluation. If the re-evaluation flag is set each individual is re-evaluated at the start of a generation. This can serve a purpose if the evaluation is dependent on the state of the system or its non-stationary environment or if it the evaluation contains stochastic elements. In many static optimisation problems the fitness of an individual is deterministically dependent on the individual and re-evaluation will serve no purpose.

7.3 Hybridization of Genetic Algorithms with the Back-Propagation Algorithm

Genetic algorithms have been used to search the weight space of a MLP network without the use of any gradient information. In this technique, a complete set of weights is coded in a binary or real number string, which has an associated fitness indicating its effectiveness. Starting with a random population of such strings, successive generations are constructed using genetic operators to construct new strings out of old ones such that fit strings are more likely to survive and to participate in crossover operations.

Due to the fact that SUGAL makes it very easy to perform multiple runs and keep a record of several statistical parameters, it was chosen to use this program for the back-propagation runs. So just like using the genetic algorithm, each neural network is retained by storing its weights on a chromosome. The number of runs is set as the population size, e.g. if a neural network had to be trained by the back-propagation ten times, each time with random initial weight values, the population size would be set to ten. All weights of each chromosome are randomly initialised in a uniform range between '-1' and '+1', as is common for the back-propagation initialisations. By uniformly selecting the two 'parent'-chromosomes, each chromosome would be selected only once to produce 'offspring'. Of course, in this case 'offspring' refers to a

'parent' neural network, which has been trained for a number of additional epochs. Since fitness normalisation is useless, this is set to 'direct' so as not to waste any time with useless computations. SUGAL always selects two parents (as is natural for a genetic algorithm), which are then passed to the routine performing crossover. Since of course crossover must not be performed in this case, the crossover rate p_c is set to 0, effectively preventing the two neural networks from being mixed. The actual back-propagation training has been programmed in a new mutation routine, called 'BPlearn', which has been added to the usual mutation routines. By setting the mutation rate p_m to 1.0 per chromosome, it has been made certain that this mutation routine was then called upon exactly once for each chromosome, before inserting them in the new population. This replacement is done 'uniform' and 'unconditional', which assures that the newly trained networks do not have to compete with other networks for a place in the new pool. Of course, 'elitism' has been turned off, since this would not only prevent one network from being trained for one generation, but also upset the order of the pool by leaving one parent unselected for training and insertion in the next pool, which would then be lost forever. Lastly, in order to reduce the overhead of the algorithm, it was chosen that the mutation operator 'BPlearn' would learn the networks for fifty epochs before inserting it in the new pool, rendering one generation in SUGAL equal to fifty back-propagation epochs. It can be seen that this way, SUGAL performs an effective simultaneous back-propagation learning algorithm for any number of networks and for any number of epochs, while collecting statistical information as the minimum and mean network error and, for instance, their standard deviation. Out of an initial random pool of networks, all networks are picked (two at a time), which are then trained individually and inserted into the pool of the next generation. This is repeated until the desired number of generations (and thus epochs) has been reached. By inspecting the pool, the exact network error of each individual network can be obtained, giving vital information about the diversity of the final results.

- **Coding**

The coding is chosen to be real-valued. A single chromosome represents all the weights in the neural network (including the bias weights), where a single real-valued gene corresponds to one weight-value. The order in which the weights are represented in the chromosome string is quite ambiguous. The nodes in the network are numbered from '0' starting at the bias-unit, then the input units, the hidden neurons and finally the output neurons. Even though the input units and the bias unit are not really neurons at all, they will be referred to as such as is common practice. The network architecture is not restricted to a classic fully connected layer-model. However, the hidden neurons are numbered in such a way that neurons with a higher index are 'higher' up in the hierarchy of the network; i.e. neurons can only have outgoing connections to neurons with a higher index. The indices of the weights represent the order in which they appear in the chromosome. Incoming weights to a certain neuron are grouped together in the chromosome representation. Example of the ordering of the weights in a chromosome is shown in Figure 7-2.

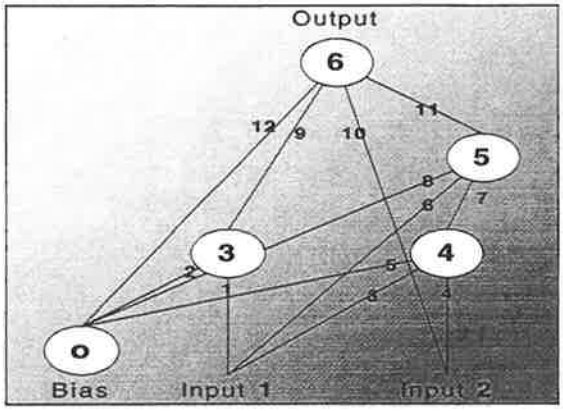


Figure 7-2: Example of the ordering of the weights in a chromosome.

7.4 Neural Network Specific Crossover Operator

In genetic algorithms, the crossover operator plays an important role. It is a very critical operator, without which there simply would not be any 'real' genetic algorithms since it is the only possible form of exchange of information between the chromosomes of a population. In general, two very important factors have to be recognised when dealing with crossover: NN-specific crossover operation is shown in Figure 7-3.

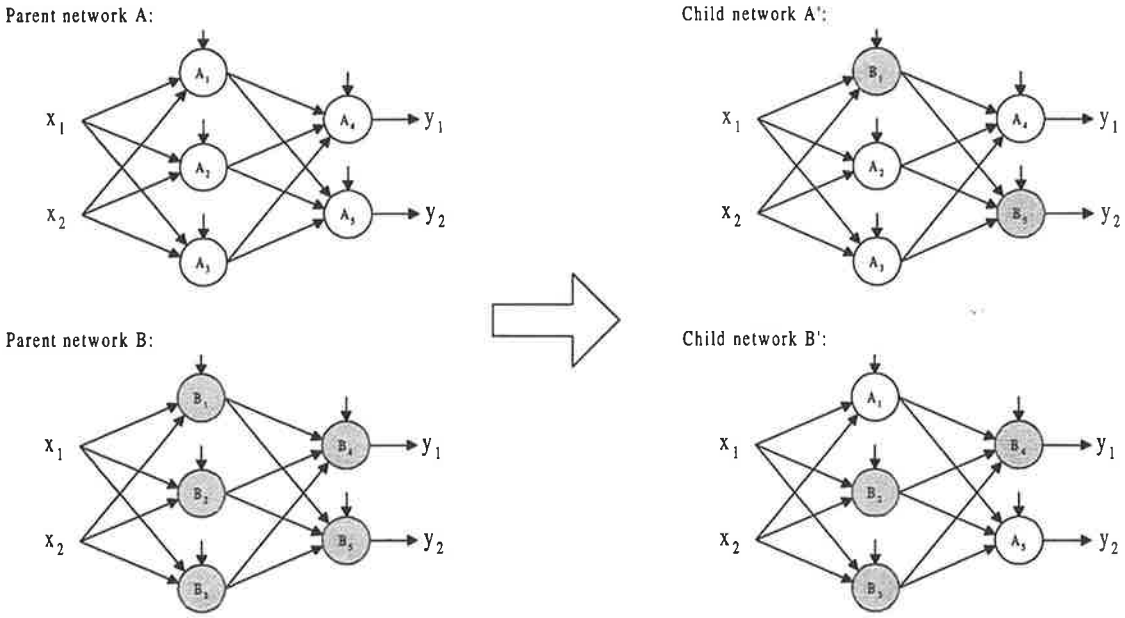


Figure 7-3: NN-specific crossover operation.

- **Constructive Crossover Operator**

Crossover is needed to combine partial information from several chromosomes into one superior solution. It combines lower order building blocks into new, higher order building blocks. This effect of crossover is essential to the functioning of the algorithm, and looks at crossover as a constructive operator.

- **Disruptive Crossover Operator**

Crossover can also very easily disrupt the building blocks. This way, crossover is seen as a disruptive operator, whose effect must be limited as much as possible. These two 'functions' of crossover form a certain contradiction, often resulting in a trade-off. This can for instance be done by either choosing multi-point (or uniform) crossover with a low crossover rate, or 2-point crossover with a higher crossover rate.

In genetic algorithms, a further difficulty arises, as the weights are very much dependant on each other. In other words, setting one weight to a better value could have no effect whatsoever unless the other weights of that particular neuron are also set to their appropriate values and making it possible for the neuron to make an effective separation of its input space (in other words: forming a useful hyperplane). Furthermore, this important relation between weights belonging to the same neuron also makes it very unlikely that selecting some weights from one chromosome and the other weights from another chromosome would result in successful neuron operation. This 'intra-crossing' of neurons has already been limited by the manner in which the cell-weights are coded onto the chromosomes, locating all weights of a neuron next to each other. However, by eliminating all possibilities of selecting a crossover point within such a weight-group, this could be taken a step further: it would be absolutely eliminated.

The NN-specific crossover operator determines the potential crossover points (as shown in Figure 7-4), which lie in between the weight groups of the various neuron cells. Then, it selects which of these points will actually be used for crossover. With this neural network specific crossover operator, previously evolved neurons can no longer be disrupted by crossover, whereas the various neurons from the available chromosomes can still be freely combined by this same operator. A uniform NN-specific crossover operator would select more points for this than for instance a 2-point NN-specific crossover operator. When the crossover operation is actually performed, two child chromosomes will be generated representing networks A' and B', which consist of neuron cells taken as a whole from the parental networks.

This new crossover operator has the disadvantage that with the potential crossover points within a neuron's weight group, also the possibility of forming new useful hyperplanes by crossover has been eliminated. However, it is expected that the advantages outweigh this disadvantage, experiments indicate that the dominant search method is genetic hill climbing, where the mutation operator is used to optimise the various weights or hyperplanes. This way, the mutation and crossover operators would gracefully work together, with mutation optimising the various neurons and crossover subsequently recombining them into better networks.

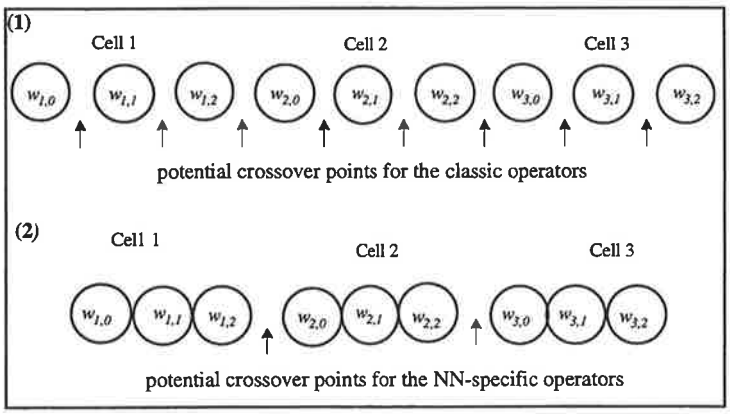


Figure 7-4: Example of the difference in potential crossover point

The neural network used to classify this data was a fully connected feed-forward neural network, with thirteen input neurons, ten hidden neurons and one output neuron. Including the standard neuron thresholds, the genetic algorithm requires chromosomes with 151 genes.

7.5 Simulation Results

For this reasonably complex problem, various settings for the learning rate η and the momentum rate μ are used: $\eta = 0.5$ with no momentum ($\mu = 0$), $\eta = 0.5$ with momentum $\mu = 0.5$, $\eta = 0.1$ with $\mu = 0.0$, and finally $\eta = 0.1$ with $\mu = 0.5$. For each of these settings 100 runs are performed, during which the networks are trained for 12,500 epochs.

It appears that the best results by far were obtained using the fourth setting ($\eta = 0.1$, $\mu = 0.5$): an average network error of 0.44 and 74% of the runs reaching a perfect classification of the whole training set. The worst run from the batch using this particular setting got stuck at an error of 2.51.

The first setting ($\eta = 0.5$, $\mu = 0$) produced only one perfect classification out of the 100 runs performed. Addition of a momentum parameter ($\mu = 5$) gave no improvement. The results for the other settings show that the learning rate η is set too high in the first two cases, making it almost impossible for the back-propagation algorithm to find solutions with low network errors. When we compare these results with those obtained by the genetic algorithm, the conclusion must be drawn that the back-propagation definitely outperforms the genetic algorithm again if the right settings for the learning rate η and momentum rate μ are used. The best solution found by the genetic algorithm still had a network error of 0.891 (NN-specific uniform crossover), after a comparable amount of training.

Again, 12500 training epochs are calculated for each of the performed 100 runs. Surprisingly, it seemed to be extremely difficult for the back-propagation algorithm to obtain a 100% perfect classification rate, since only one run has succeeded in doing so (this run can be seen as 'best' run in the graph). The average network error reached a value of 4.0 after about 2000 training epochs, after which it very gradually descends to a final value of 3.75 after the full 12500 steps. Figure 7-9 shows how the training errors of the various networks are spread out. It appears that 69% of the networks has a final

error between 2 and 5, and 18 networks perform even worse than that, with errors ranging up to 10.4.

It seems that this real-life medical problem is very hard for the back-propagation to optimise, given the fact that only one run out of a hundred could find a perfect solution, whereas the others seemed to have got stuck in a local minimum. This points towards a very complex error surface with a very high probability of the algorithm getting stuck before reaching a more global optimum.

When comparing these results with those obtained by the genetic algorithms, it can be concluded that, for this problem, the genetic version outperforms the back-propagation. The NN-specific crossover types got an average final network error of 2.91 and 3.13 ($p_c = 1.0$), the back-propagation obtained an average of 3.75 after a roughly comparable amount of calculations. Of course, these results were obtained with the back-propagation algorithms without momentum, but it still shows that genetic algorithms can actually get good and competitive results. The genetic algorithms do take a lot of time, about 250 to 300 generations, before they get near a network error of around 4, compared to 2000 training epochs for the BP-algorithm, which is comparable to only 80 GA-generations. But the probability of getting stuck in a local optimum is much smaller than for the back-propagation, since the mutation operator provides a powerful tool to escape these traps again. This could lead to a situation as is observed here, where back-propagation initially takes a strong lead, but further down the track often finds itself stuck, whereas the genetic algorithm can continue in slowly but steadily decreasing the network error. A necessity for this typical situation is a certain type of error surface, which makes it hard for the back-propagation to find a solution.

In order to see if the use of a momentum parameter in the back-propagation routine would give an improvement, ten runs were performed, again for 12500 epochs, with the learning rate ε set to 0.1 and the momentum μ set to 0.9. Indeed, this resulted in a better performance, with two out of ten runs reaching a perfect solution, with a final network error of around 0.0015. Three runs ended with an error between 1 and 2, another three with an error between 2 and 3, and finally two runs ended with an error between 3 and 4. The total average training error was 1.71, which is undoubtedly better than both the back-propagation without momentum and the genetic algorithms. An average training error of 3.0 was already reached after as little as 200 epochs, and the best run reached a perfect classification rate within 150 training steps. Again, this shows that the back-propagation can be extremely fast, if it is lucky enough to begin with a good initial starting point.

Again, every setting is run 25 times, each time with different random initialisations. The network errors of the best chromosome of each generation are accumulated, averaged and plotted. The results obtained by the various crossover operators after the 500 generations are presented in Table 7-1.

Table 7-1: Simulation results for the CAD problem after 500 generations.

	Average of the runs			**Geometric Mean of the runs**		
Crossover type	**$p_c = 0.6$**	**$P_c = 0.8$**	**$p_c = 1.0$**	**$p_c = 0.6$**	**$p_c = 0.8$**	**$p_c = 1.0$**
1-point	3.616	3.683	3.279	3.501	3.581	3.133
2-point	4.116	3.750	3.404	4.028	3.582	3.213
Uniform	4.077	3.812	3.492	3.950	3.613	3.307
NN-2-point	3.555	3.279	2.907	3.345	3.089	2.702
NN-uniform	3.276	3.205	3.129	3.147	2.950	2.902

Again, the resulting network errors indicate that many of the runs did not reach a perfect classification of all the input vectors. In contrast to the previous problem, this time the geometric mean values are also not reduced below values of around three to four. And although the graphs indicate that the genetic algorithms keep on reducing the network error, such perfect solutions are not likely to be found within many generations to come. In fact, the lowest network errors found by each crossover type in all its runs are 1.001 for 1-point crossover, 0.962 for 2-point crossover, 1.000 for uniform crossover, 0.891 for NN-specific uniform crossover and finally 0.998 for NN-specific 2-point crossover. These values are almost equal, but almost certainly still not low enough to guarantee a 100 % classification. It can be estimated with reasonable accuracy that, after 500 generations, on average 3 training vectors is still misclassified, which equals a correct classification rate of 97.5 %.

The simulation results improve as the crossover rate increases from 0.6 to 1.0. Strangely enough, 1-point crossover performs best for the classic versions. For the NN-specific types, the uniform version performs better for $p_c = 0.6$, equal results are obtained for $p_c = 0.8$ and the 2-point version performs better for the highest crossover rate. This means that for the first time uniform NN-specific crossover does not always give the best results. Again, it is clear that the NN-specific operators outperform the normal crossover operators, although the difference is not very great.

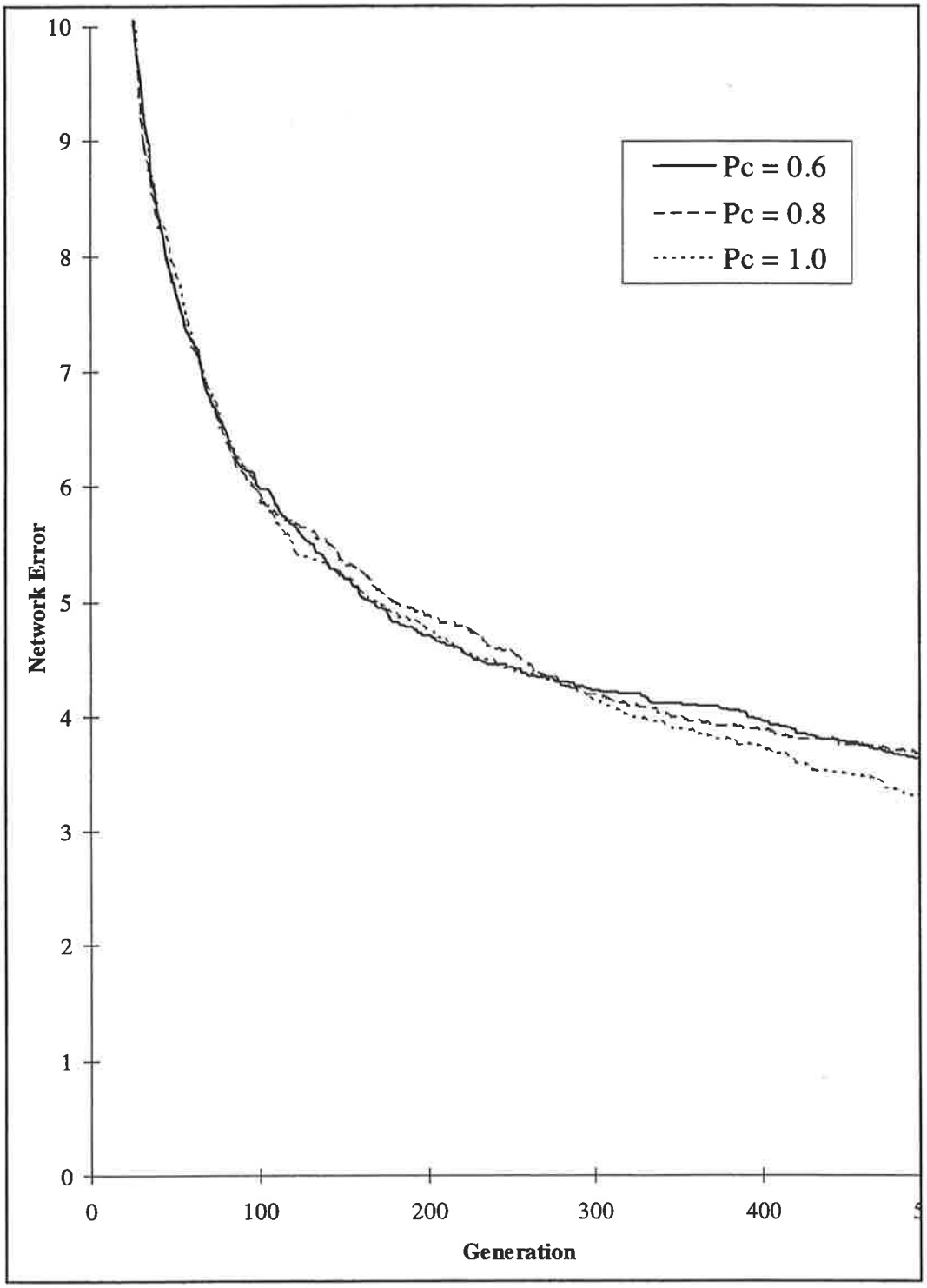


Figure 7-5: 1-point crossover averaged over 25 epochs

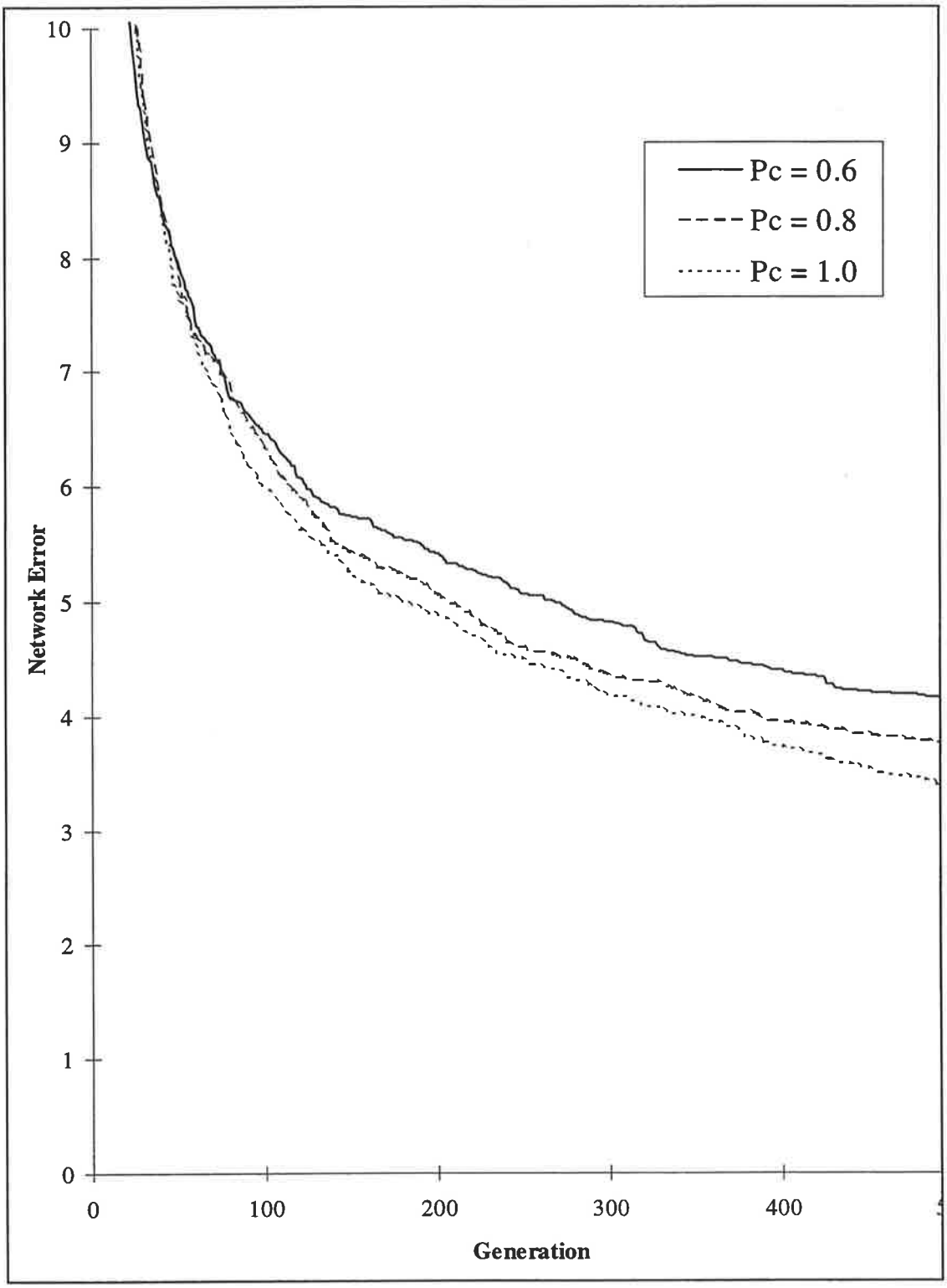


Figure 7-6: 2-point crossover averaged over 25 epochs

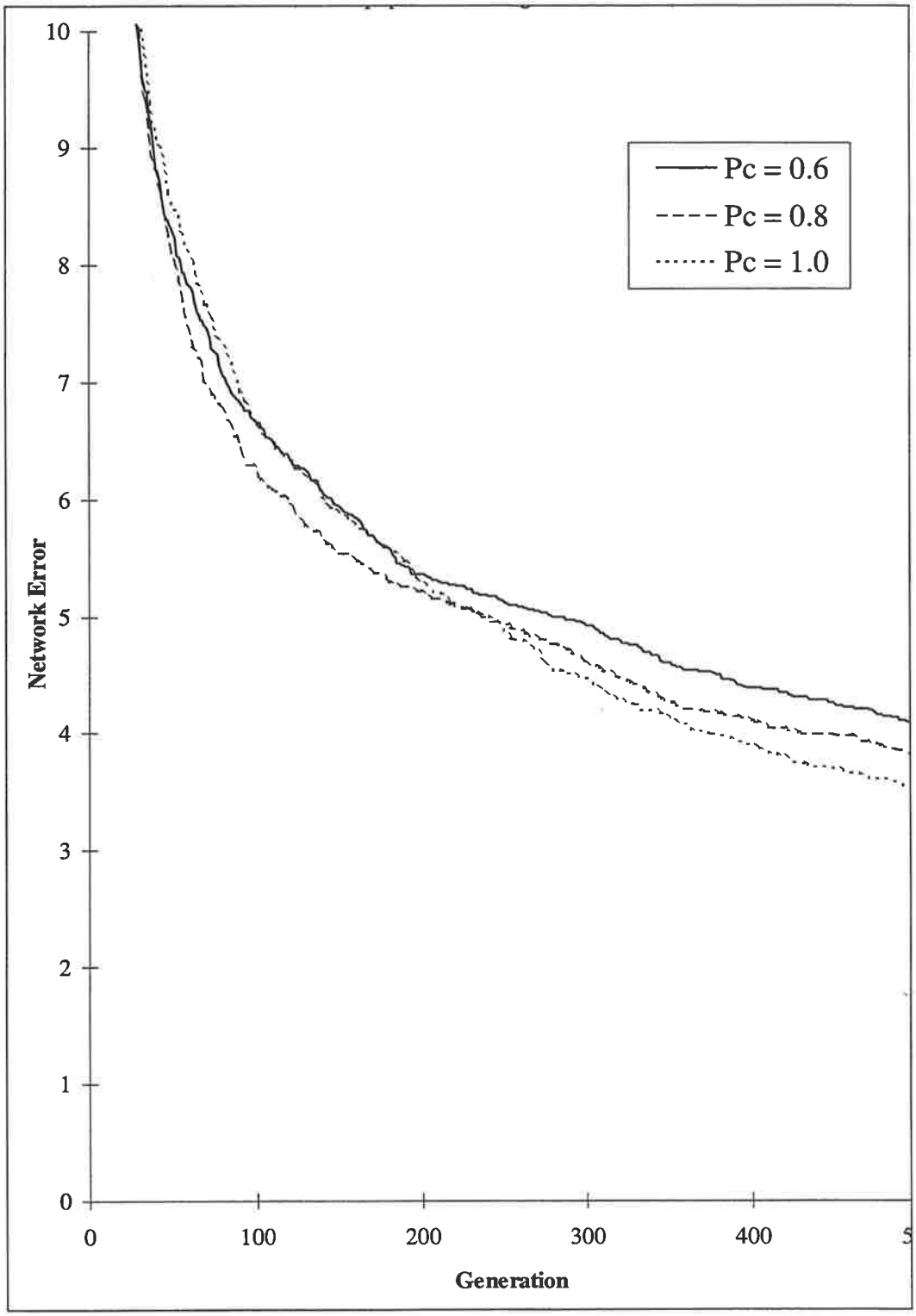


Figure 7-7: Uniform crossover averaged over 25 epochs

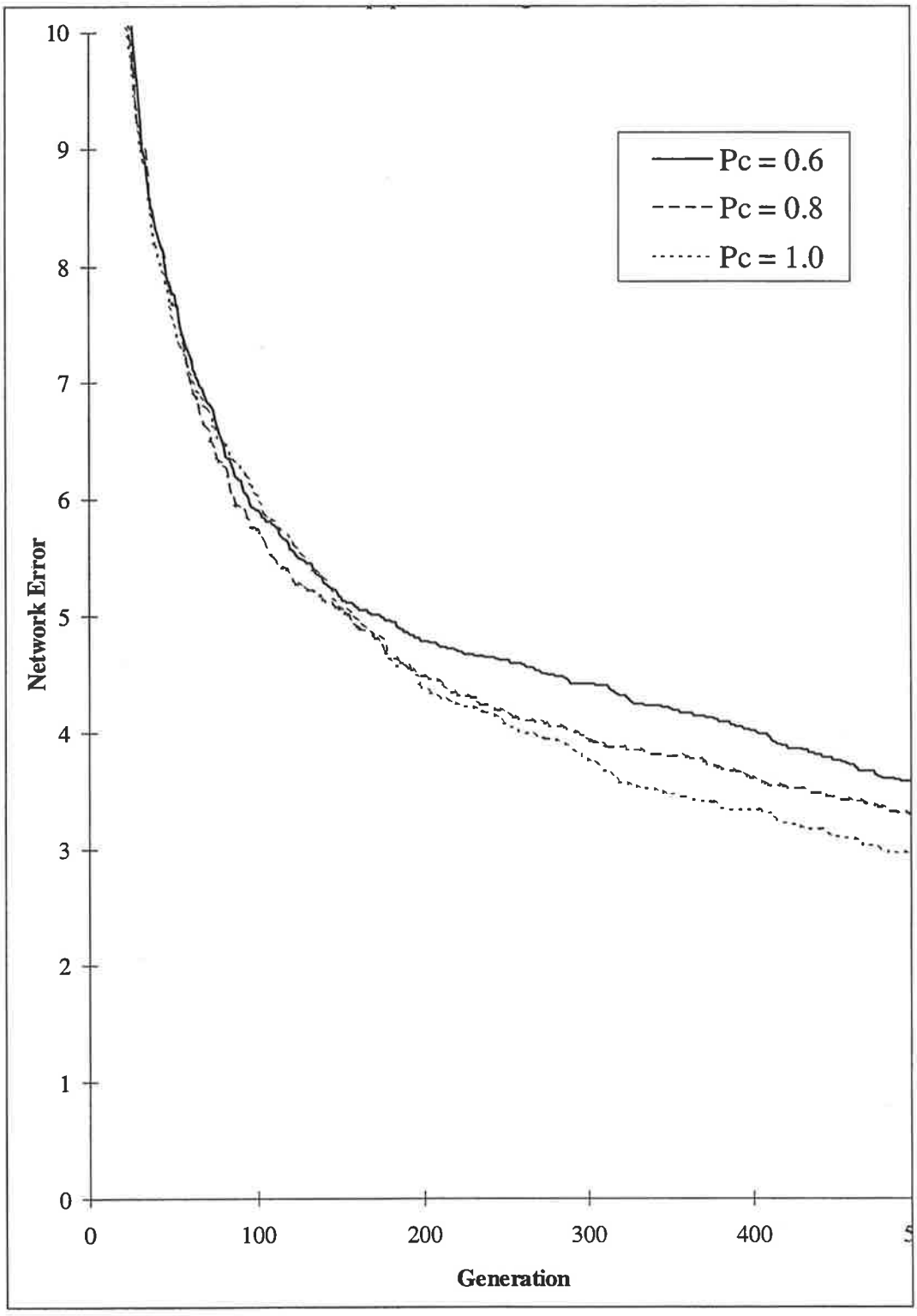


Figure 7-8: NN-specific 2-point crossover averaged over 25 epochs

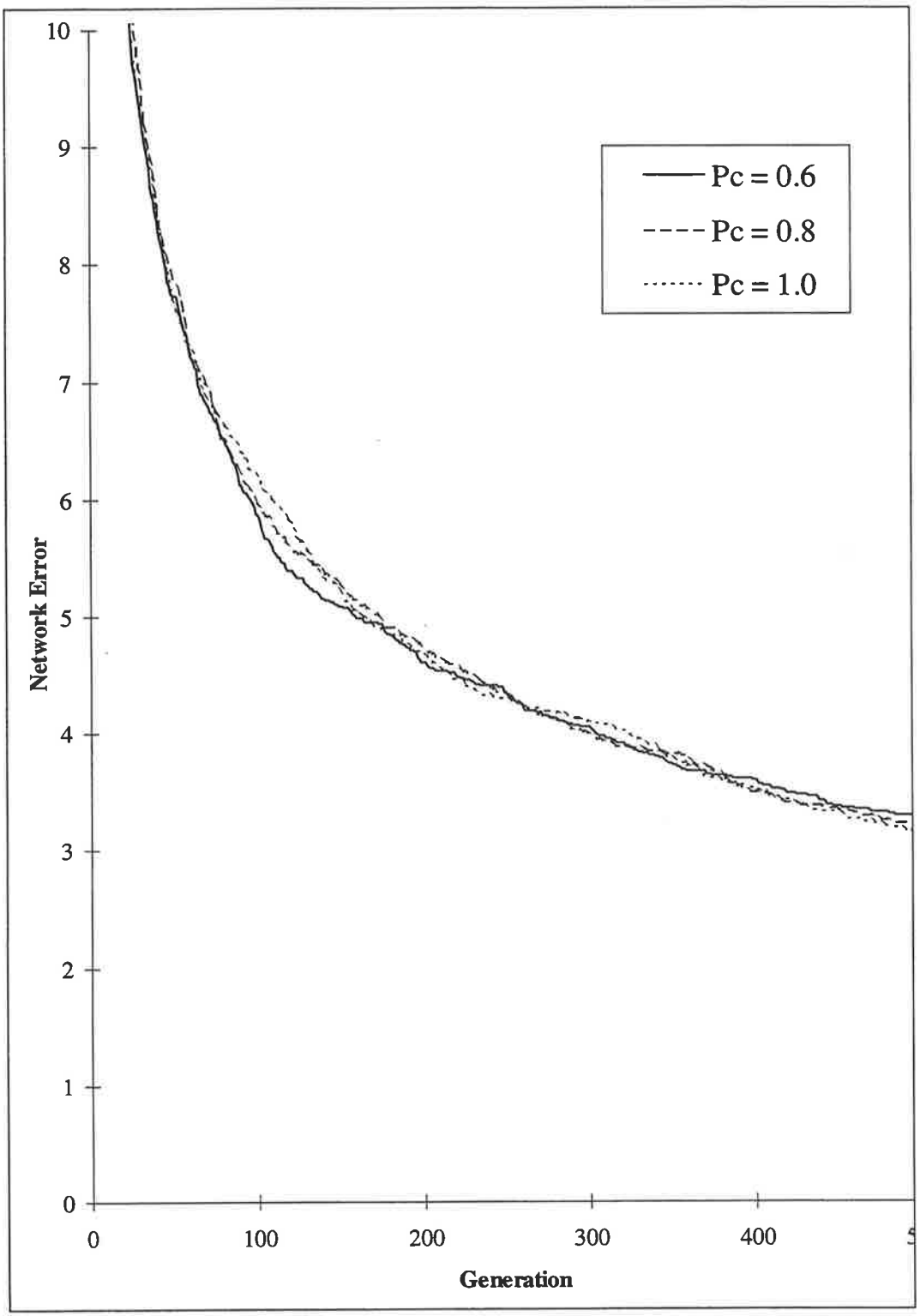


Figure 7-9: NN-specific uniform crossover averaged over 25 epochs

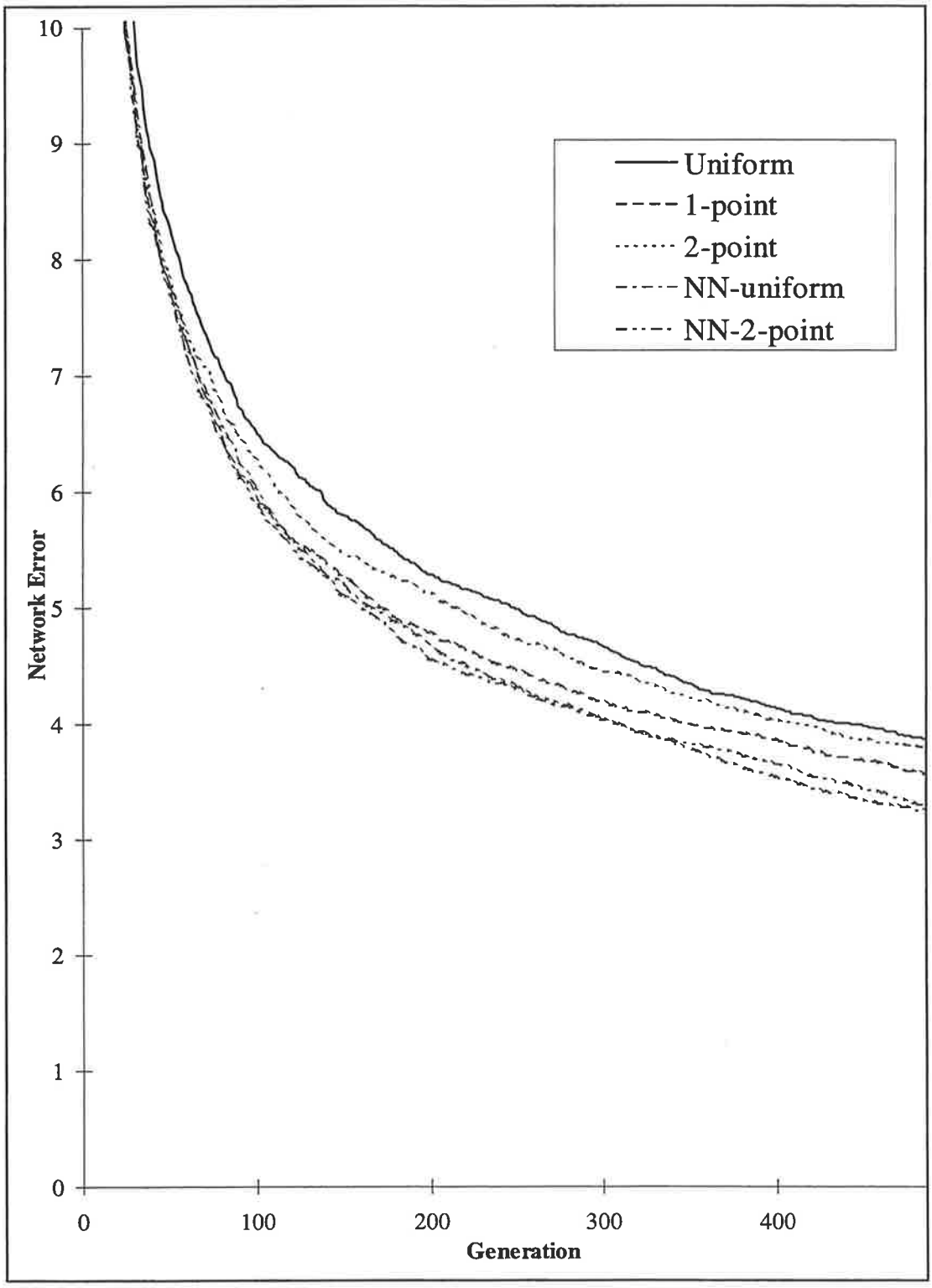


Figure 7-10: Average of 75 epochs

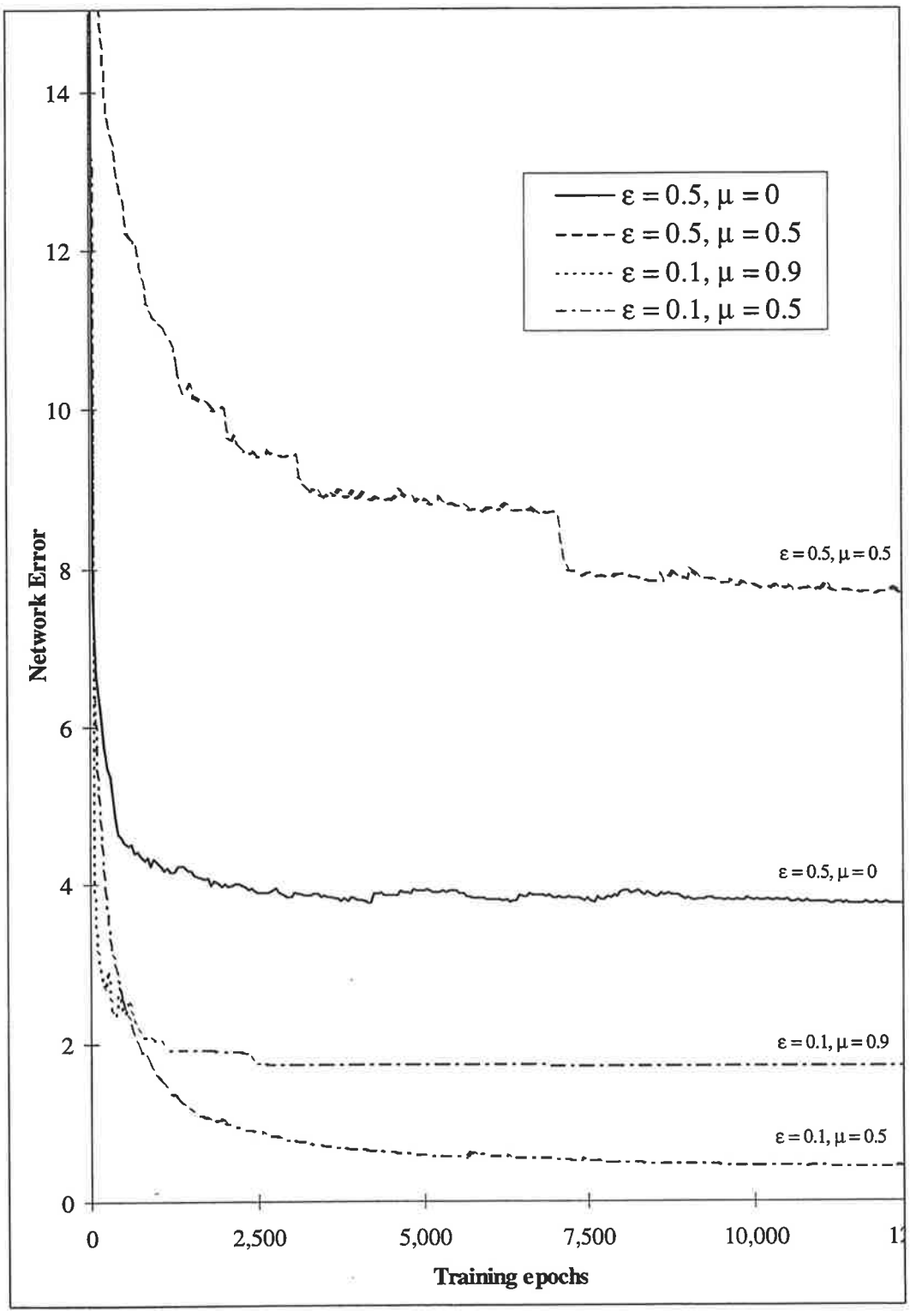


Figure 7-11: Averages over 100 epochs

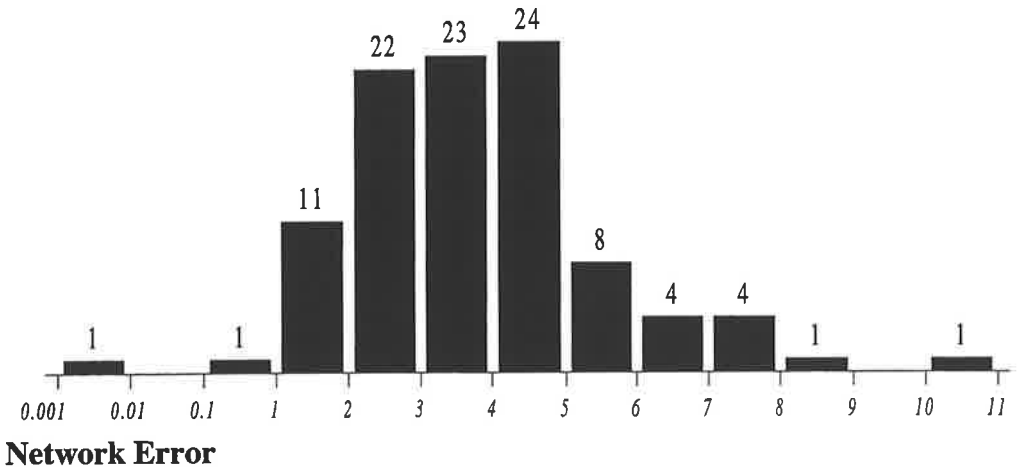


Figure 7-12: No. of networks with an error within various ranges

7.6 Conclusion

In this chapter it is shown that genetic algorithms can be used to optimise network weights. The main advantage of using GAs for the training of MLP is that they can find global minima without getting stuck at local minima. The basic difference between BP and GA based training mechanisms is that, unlike BP, GA does not make use of local knowledge of the parameter space. Experiments indicate that the genetic version definitely outperforms the back-propagation if the learning rate and momentum are chosen correctly. Several experiments have been carried out on a coronary artery disease data set with various crossover operators, 1 point, 2 point, and NN uniform, NN 2 point on this data set. The lowest network errors found by each crossover type in all its runs are 1.001 for 1-point crossover, 0.962 for 2-point crossover, 1.000 for uniform crossover, 0.891 for NN-specific uniform crossover and finally 0.998 for NN-specific 2-point crossover. It has been found that, after 500 generations, on average 3 training vectors were misclassified, which equals a correct classification rate of 97.5 %.

Chapter 8

Experimental results using genetic programming

8.1 Introduction

In this chapter the performance of the *Evolutionary Pre-Processor* (EPP), a genetic-programming-based feature classifier is examined. Genetic programming (GP) is analogous to a genetic algorithm, except that the structures undergoing adaptation are trees rather than strings. The trees are hierarchical representations of computer programs or functional expressions. The internal nodes are functions that present their output to their parent node, and take the outputs of their child nodes as arguments. The leaf nodes are terminals which are the inputs to the program. The output of the program is taken from the root (top) node. The set of functions and terminals are defined by the user, and are specific to the problem. Functions can have side effects such as altering the state of memory. In order to apply GP to a problem, there are five things that must be specified by the user (Koza, 1992b):

1. the set of terminals
2. the set of functions
3. the fitness measure
4. the parameters for controlling the run, and
5. the method for designating the result and the criterion for terminating a run

The Evolutionary Pre-Processor, is an automatic non-parametric method for the extraction of non-linear features. Using genetic programming, the Evolutionary Pre-Processor evolves networks of different non-linear functions, which preprocess the data to improve the discriminatory performance of a classifier. The performance of each approach for test patterns, that is the generalisation ability of each approach is evaluated by cross validation techniques. Simulation results are compared with the performance of various classification methods such as k-Nearest Neighours (knn), hyperplane, linear perceptron and Gaussian Maximum Likelihood. All experiments were performed on a real world-data set of coronary artery disease. The results are compared to those of several other classification methods on the same problem. Control parameters required for standard GP are shown Table 8-1.

Table 8-1: Control parameters required for standard GP.

1. population size
2. max. number of generations
3. probability of crossover
4. probability of mutation
5. probability of reproduction
6. probability of choosing internal points for crossover
7. selection scheme
8. fitness scaling scheme
9. max. depth of trees created during a run
10. max. depth initial random trees
11. method for generating initial population
12. elitist strategy

8.2 The Evolutionary Pre-Processor

The Evolutionary Pre-Processor is a new method for the automatic extraction of non-linear features for supervised classification. The central engine of Evolutionary Pre-Processor is the genetic program; each individual in the population represents a pre-processor network and a standard classification algorithm. Based on genetic programming the Evolutionary Pre-Processor maintains a population of individuals, each of which consists of an array of features. The features are transformations made up of functions selected by the user. A fitness value is assigned to each individual which quantifies its ability to classify the data. This fitness value is based on the ability of a simple classifier to correctly classify the data after it has been transformed to the individual's feature space. Through the repeated application of recombination and mutation operators to the fitter members of the population, the ability of the individuals to classify the data gradually improves until a satisfactory point in the optimisation process is reached, and a solution is obtained.

The Evolutionary Pre-Processor performs a search over combinations of different, complementary components to solve a classification problem. The Evolutionary Pre-Processor non-linearly transforms the original data measurements to a set of features, which are then passed to the specified classification algorithm. Each individual can therefore be considered as a pattern classifier. The objective of evolution is to improve the classification accuracy of the individuals on the training data.

Using genetic programming, Evolutionary Pre-Processor:

- automatically selects the number of features to extract.
- uses non-linear pre-processing functions which are appropriate for the problem.
- performs feature selection by determining which of the original measurements are useful for classification.
- searches for the most appropriate classification algorithm from among a given set.
- evolves the size and structure of the pre-processing network.

Some of the main differences from standard genetic programming are:

strong typing: the mutation and random creation of individuals is constrained such that the output type of each terminal or function matches the argument type of its parent. The three types used in Evolutionary Pre-Processor are real, enumerated and Boolean.

dynamic trial allocation: the computational complexity of Evolutionary Pre-Processor is enormous: each data element must be processed for each individual, for each generation, for each run. To reduce computation time, only a subset of the training samples are used to evaluate fitness. The critical number of samples needed to determine the rank of each individual with statistical confidence is determined through the Rational Allocation of Trials (RAT).

genetic operators: the current view in evolutionary computation is that no single genetic operator is optimal for every problem. Therefore Evolutionary Pre-Processor uses crossover and up to seven mutation operators which are applied probabilistically. Each individual contains a vector of operator probabilities so that they can be evolved along with the population. The operators function at two levels: crossover occurs at a high level, such that the function trees are treated as indivisible genes, whereas mutation occurs at the low level, operating on the individual features.

local optimisation: genetic programming is notoriously bad at evolving numerical constants. Evolutionary Pre-Processor includes a local optimisation step during which the constants in each individual are optimised. The real-valued constants are modified using a simplex algorithm, while enumerated constants are manipulated using simple hill-climbing.

Experiments

The methods applied in the experiments are:
Evolutionary Pre-Processor: the following functions set was used: {+, -, ×, /, =, <, If-then-else, And, Or, Not}.

Multi-layer Perceptron (MLP): trained using the RPROP algorithm. Thirty-two different architectures were examined and that with the lowest validation set error was selected as the best.

Quest: this is a decision tree algorithm based on the FACT method which makes an un-biased selection of variables when forming. The default parameter values were used, with univariate splits and cross-validation pruning.

k-Nearest Neighbours (kNN): the training samples nearest to the incoming sample decide its class by majority vote.

Gaussian Maximum likelihood (ML): a Gaussian distribution is assumed for each class, and a separate mean and covariance matrix is estimated for each class, resulting in quadratic decision boundaries.

Linear Perceptron (GLIM): the feature space is segmented by hyper-planes which are trained using an error-correcting.

Parallelepiped (PPD): the extents of the samples belonging to each class are calculated in each dimension to establish a bounding box around each class. A new sample is classified according to the parallelepiped it lies in; if it lies in more than one or outside all the boxes, the box with the nearest centre is chosen.

Minimum-Distance-to-Means (MDTM): the new sample is assigned to the class with the closest mean.

Evolutionary Pre-Processor and the MLP are both stochastic algorithms; the rest are deterministic. The kNN algorithm has a hyper-parameter which was selected by minimisation of the validation set classification error over the range k = 1...50. The most important aspect of a classifier is its ability to generalise: that is, to accurately predict the class of a previously unseen object. In order to test generalisation, the data were partitioned into three sets of samples: the training set, the validation set and the test set. Both the training and validation sets were used to derive the final classifier, while the test set was put aside to obtain a statistically independent estimate of classification error. Some of the classification methods have a model-selection step. The training set was used by the classifier to learn the model parameters, while the validation set was used to select the best model. For those methods which yield only a single model, the training and validation sets were merged to form a single training set. A partition of 50 training, 25 validation and 25 test (460 training samples, 260 validation, 260 test) was used in all experiments.

Table 8-2: Percentage classification errors for each of the methods used.

Classification Error		EPP	MLP	Quest	kNN	ML	GLIM	PPD	MDTM
Heart_1	training	19.13	18.04	-	17.97	17.97	35.51	43.62	40.34
	validation	18.70	16.09	22.32	-	-	-	-	-
	test	21.74	18.70	19.36	19.13	19.13	53.04	47.83	45.65
Heart_2	training	22.39	20.00	-	18.12	18.12	35.07	49.71	41.16
	validation	15.65	13.04	22.65	-	-	-	-	-
	test	23.48	19.13	23.56	17.83	17.83	26.96	53.91	41.74
Heart_3	training	17.17	24.35	-	16.09	16.09	35.80	51.30	41.74
	validation	16.52	16.09	23.29	-	-	-	-	-
	test	16.96	20.00	26.71	20.87	20.87	39.13	52.17	39.57

There are several sources of variation, which were addressed in the experiments:

Partition of the data: often the number of samples available for classification is sufficiently small that the results are significantly affected by the partitioning of the data into the training, validation and test sets. Results are presented for three random partitions of the data set, referred to as heart_1, heart_2 and heart_3. Although permuted and partitioned randomly, the samples were selected in such a way that the population class proportions were preserved within each data subset.

Permutation of the data: some methods, such as the MLP, are sensitive to the ordering of samples in the training set. This is dealt with by the multiple partitions of the data.

Random Initial Conditions: Evolutionary Pre-Processor and the MLP both rely on random initial conditions. Each was run 10 times on the three permutations of the data for more reliable results. In each case, the run which yielded the minimum validation set error was chosen as the best run to obtain the result.
Missing attributes were dealt with in the following way: for real-valued variables, a missing value was replaced with the out-of-range value -9; for enumerated variables, an extra enumeration value was added called ``missing".

8.3 Results

Table 8.1 shows the training, validation and test set classification errors as a percentage of that set for each of the three permutations of the data set. The test set error is the most important, since it gives a statistically independent estimate of the error rate of the classifier. For the ML, GLIM, PPD and MDTM classifiers there is no model selection mechanism and therefore no validation set error. Since Quest uses 10-fold cross-validation, it does not have a training set error. The Evolutionary Pre-Processor training set error can be deceptive because it is based only on a subset of the training data which is selected according to the RAT algorithm.

The results show that the simple classifiers kNN, GLIM, PPD and MDTM performed poorly in all cases. The ML algorithm, also a fairly straight-forward method, performed surprisingly well. It appears the quadratic decision boundaries were fortuitously appropriate for this data set, since the presence of discrete data nullifies the assumption of normally-distributed data. The accuracies of the Evolutionary Pre-Processor, MLP, Quest and ML algorithms were all generally good, except for the poor performance of Quest on the third permutation. The MLP and ML classifiers had the most robust performance over the three permutations, but no algorithm had superior performance over all permutations.

There are other criteria for comparing the classifiers, such as computation time and understandability of results. The simpler methods ML, Perceptron, PPD and MDTM took on the order of seconds to complete their task. Quest is about one order of magnitude slower, but still quite fast. Evolutionary and the MLP took about three orders of magnitude longer than the simpler methods.
Figure 8-1 shows the solutions generated by Evolutionary Pre-Processor and Quest for heart_2. The other methods were not considered to yield any intuitively interpretable information about the problem. The trees generated by Evolutionary Pre-Processor shown in represent parse trees for functional expressions: each internal node is a function which takes its child nodes as arguments and returns the result to its parent.

Figure 8-2 show the decision tree obtained by Quest; when true, the splitting conditions indicate that the left branch be traversed. Both Evolutionary Pre-Processor and Quest yield possibly useful information about the problem domain, and both indicate which of the attributes are useful for discrimination purposes. Evolutionary Pre-Processor has the advantage that it is able to generate more flexible rules than Quest, but they are generally more difficult to interpret since the features are then passed to a classifier.

Genetic programming usually results in spurious subtrees which do not contribute to overall fitness but have hitch-hiked along with fitter portions of code. These unhelpful nodes make the results more difficult to interpret, and can be misleading. There were nevertheless some superficial similarities between the features generated by Evolutionary Pre-Processor for the three permutations of the data, such as the prevalence of the feature X3 = asymptomatic, and the association between X10 and X8, X4 and X1. The trees generated by Quest seem more consistent between the different permutations, although the tree for heart_3 is somewhat disappointing, containing only one split. There are few similarities between the results of Evolutionary Pre-Processor and Quest on heart_2, other than the use of X3. It is interesting that X6 was not used by Evolutionary Pre-Processor but was used by Quest, and that Evolutionary Pre-Processor used some variables which were not required by Quest.

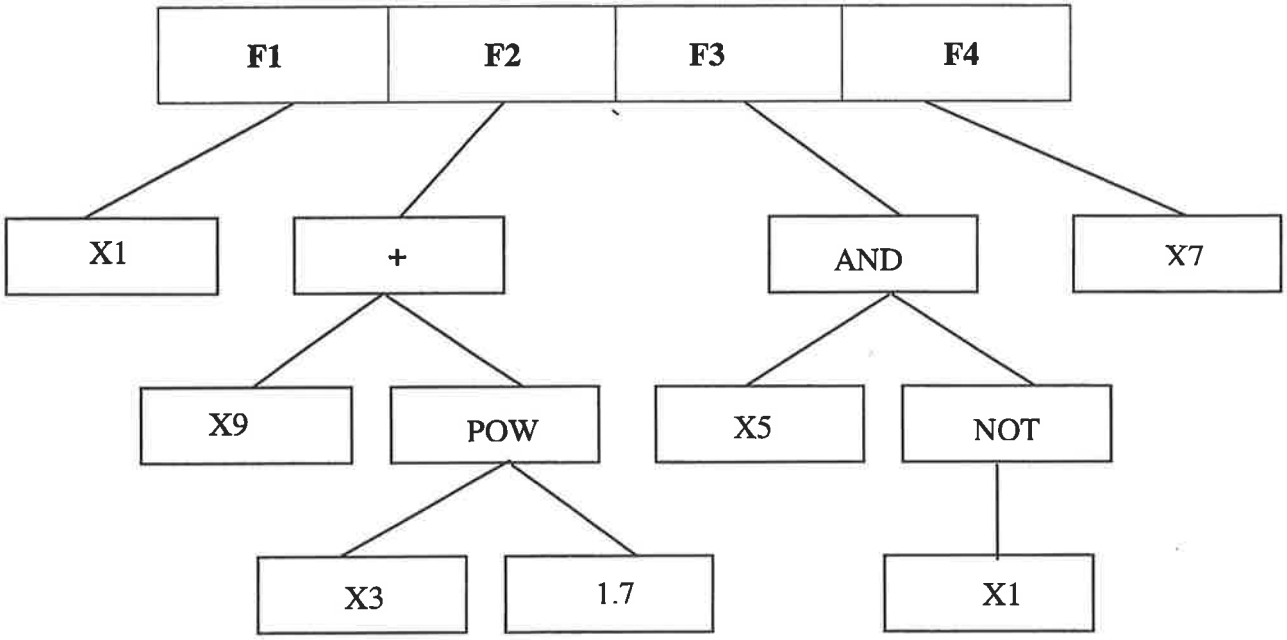


Figure 8-1: The multi-tree representation.

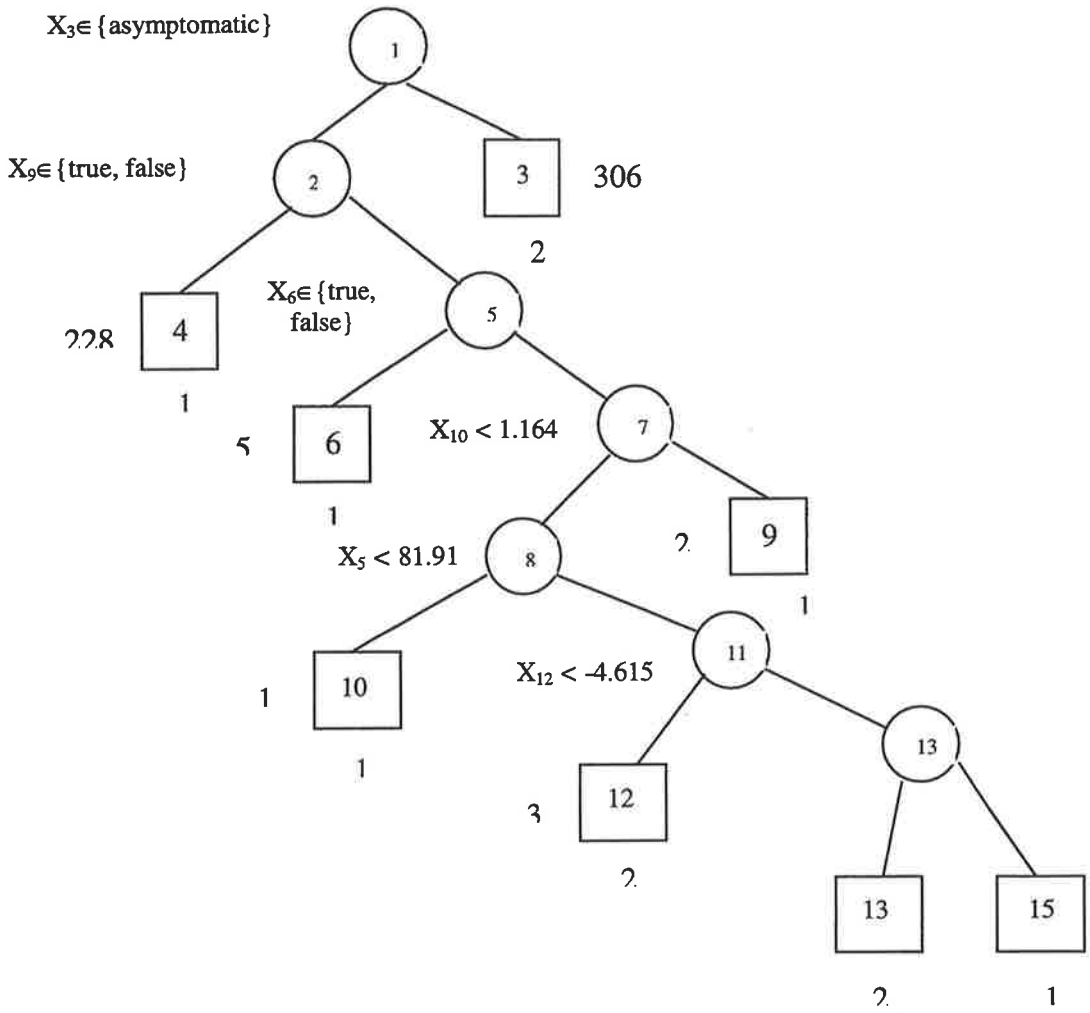


Figure 8-2: Decision Tree Generated By Quest.

8.4 Conclusion

The Evolutionary Pre-Processor has been tested with a coronary artery disease data problem and compared with seven other classification methods. While the simpler half of the methods performed poorly, the Evolutionary Pre-Processor maintained relatively good accuracy while providing interpretable results. The Quest algorithm also provides interpretable results which seem easier to understand than those of Evolutionary Pre-Processor, but was more sensitive to the permutation of the data. Given the amount of missing data and the relatively small number of records available, it is difficult to come to any conclusions about the underlying mechanics of the problem. Resampling methods such as the bootstrap could provide more reliable models of the data. In parting, the reader should note that any comparison between algorithms is only relevant for the data used, and should not be generalised to other problem domains.

Chapter 9

Experimental Results of the Fuzzy-Classifier System

9.1 Introduction

In this chapter the fuzzy classifier systems that can automatically generate fuzzy rules from training patterns is used for pattern classification problems. Classifiers in this approach are fuzzy if-then rules. It can also find a compact rule set by attaching large fitness values to such fuzzy rules that can correctly classify many training patterns. That is only those fuzzy if then rules with large fitness values are selected to construct a compact fuzzy system with high classification performance.

Here each fuzzy if-then rule was treated as an individual. The fitness value of each fuzzy if-then was determined by the numbers of correctly and wrongly classified training patterns by that rule. It is shown that fuzzy rules outperformed ordinary fuzzy rules. The classification power of on real world data of coronary artery disease is shown by computer simulation.

Recently several Neural-Network-Based and Genetic-Algorithms-Based classifiers have been proposed for generating fuzzy if-then rule for pattern classification problems. Uebele (1995) have proposed a Neural-Network-Based fuzzy classifier. In this classifier separation hyperplanes for classes are extracted from a trained neural network. Then for each class, shifting these hyperplanes in parallel using the training data set for the classes approximates convex existence regions in the input space. These studies not only claim that these classifiers are superior then NN classifier but also timesaving. For example, Shigeo and Lan (1995) extracted fuzzy if then rules directly from numerical data. In their method rules were extracted from numerical data by recursively resolving overlaps between classes. Shigeo, (1997) have then proposed a fuzzy classifier with ellipsoidal regions, which has a learning capability. In this classifier each class is divided into several clusters. Then for each cluster a fuzzy rule with an ellipsoidal region around a cluster center is defined. Then the center and the covariance matrix of the ellipsoidal region for the cluster are calculated. Ishibuchi et al., (1995) proposed a generation method of fuzzy classification rules. This method is based on (1) fuzzy partition of a pattern space into fuzzy subspaces and (2) Identification of fuzzy rule for each fuzzy subspace. Fuzzy logic has gained increased attention to solve real problems, in recent years. It provides decision-making capabilities in the presence of uncertainty and imprecision. Watanabke et al., (1994) used fuzzy discrimination analysis for diagnosis of valvular heart disease. Cios et al., (1991) used an image processing technique based on fuzzy set theory for diagnosing coronary artery stenosis. They have reported a rate of true positive diagnosis of 81% while maintaining a rate of false positive diagnosis at the low level of 10%. Their results of the experiments are

very promising. In this classifier each fuzzy if-then rule is treated as an individual that is as a classifier and each population consists of a certain number of fuzzy if-then rules.

In this chapter method a genetic-algorithm-based fuzzy method is used which provides robust performance of fuzzy rules. In this method each fuzzy if-then rule is coded as an individual which is a classifier. The fitness value of each fuzzy if-then rule was determined by the numbers of correctly and incorrectly classified training patterns by that rule. This method consists of the following steps. Fuzzy partition of a pattern space into fuzzy subspaces and identification of fuzzy rule for each fuzzy subspace.

9.2 Outline of the Fuzzy Classifier System

/*Fuzzy Classifier System */

1. **Generate an initial population of fuzzy if-then Rules**
2. **Learning of fuzzy if-then Rules in the Current population**
3. **Determine fitness value of each fuzzy if then rule in the current population**
4. **Generate new fuzzy if then rule by genetic operations for the next population**
5. **Termination test. If the algorithm is not terminated steps 2 to 5 are repeated**

$$\mu_i^P = \max\left\{1 - \frac{\left|x - \alpha_i^P\right|}{\beta^P}, 0\right\}, \quad i = 1,2,\ldots,P$$

where, (9.1)

$$\alpha_i^P = \frac{i-1}{P-1}, \quad i = 1,2,\ldots,P$$

$$\beta^P = \frac{1}{P-1}$$

In this procedure the pattern space is partitioned into P fuzzy subsets and determination of fuzzy partition L, depends on the desirable rate of correctly classified patterns. The performance of fuzzy classification system based on fuzzy if then rules entirely depend on the choice of a fuzzy partition. Therefore the choice of a fuzzy partition is very important. In this method the fuzzy rules corresponding to various fuzzy partitions are simultaneously utilised in fuzzy inference. Each fuzzy subset is defined by the symmetric triangle-shaped membership function. Figure 9-1 shows generalised bell-shaped and triangular membership function. Generalised various bell-shaped membership functions are shown in Figure 9-2.

Though we can use other types of membership functions for example, trapezoid-shaped. The simplest is the triangular membership function, which is a collection of three points. The following symmetric triangle-shaped membership function

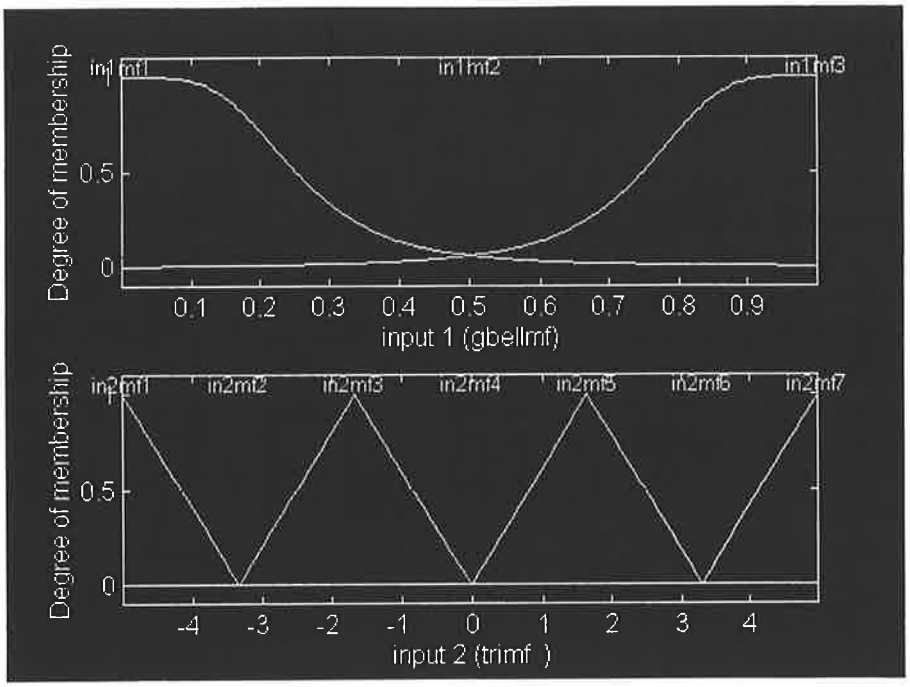


Figure 9-1: Generalised bell-shaped and triangular membership function.

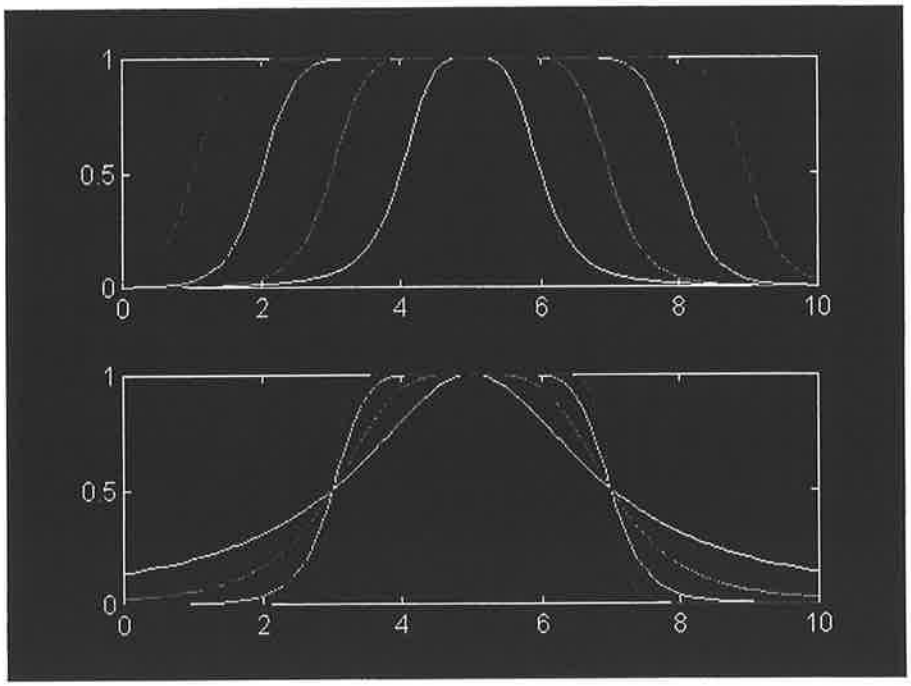


Figure 9-2: Generalised various bell-shaped membership functions.

In this membership function α is the center where the grade of membership function is equal to 1 and β is the spread of membership function.

(1) Generation of fuzzy rules

Fuzzy rule based classification method consists of the two procedures: a fuzzy rule generation procedure and a classification procedure. Suppose that Φ patterns ξ_p, p=1,2, ... , Φ are given as the training pattern in the pattern space $[0,1]^2$ from M classes. That is ξ_p belongs to one of the M classes: $\Omega_1, \Omega_2, \ldots, \Omega_M$. In this method the pattern space is divided into given number of classes to generate fuzzy rules. Each axis of the pattern space is evenly partitioned into fuzzy subsets and each pattern belongs to one of the given classes. It is important to determine appropriate fuzzy partitions of pattern space since the performance of the derived fuzzy rules is affected by the fuzzy partition. The following fuzzy rule for 2-group classification problem with grade of certainty is used. First find sum of the product of membership value of each pattern in each class, that is,

$$\gamma_{\xi t} = \sum_{p \in \xi t} \mu_{i1}^{P}(\xi_{1p}).\mu_{i2}^{P}(\xi_{2p}). \ldots \mu_{in}^{P}(\xi_{np}), \quad t = 1, 2, \ldots, M \tag{9.2}$$

Find the class ξ_x which satisfy the following condition. If there are two or more classes take the maximum value then no fuzzy rule will be generated.

$$\gamma_{\xi x} = \mathrm{Max}[\gamma_{\xi 1}, \gamma_{\xi 2}, \ldots, \gamma_{\xi m}] \tag{9.3}$$

If two or more classes take the maximum value, the consequent of the rule can not be determined uniquely.

The grade of certainty of the fuzzy rule is determined as follows.

$$GC_{ij}^{K} = \frac{\left|\gamma_{\xi x} - \gamma\right|}{\sum_{t=1}^{M} \gamma_{\xi t}} \tag{9.4}$$

where, (9.5)

$$\gamma = \sum_{\gamma t \neq \gamma x} \frac{\gamma_{\xi t}}{M-1}$$

This method can be used for generating initial fuzzy if-then rules in the fuzzy classifier system. The learning algorithm of the fuzzy classifier system can adjust the grade of certainty. In this method, the consequent is determined as the class which has the larger sum of membership value to the IF part of the fuzzy rule. The grade of certainty takes a value in the interval (0,1].

9.3 Fuzzy inference for pattern classification

Calculate for t = 1, 2, . . . , M,

$$\Psi_{\Omega t} = \text{Max}\,[\mu^{P}_{i1}(\xi_{1p}).\mu^{P}_{i2}(\xi_{1p}). \ldots \mu^{P}_{in}(\xi_{np}).GC^{P}_{ij} \mid .GC^{P}_{ij} = \Omega t;$$

i, j = 1, . . . , P, P = 2, 3, . . . , L]

Find the class Ω_x

$$\Psi_{\xi x} = \text{Max}\,[\Psi_1, \Psi_2, \ldots, \Psi_{\Omega M}].$$

$$\text{Grad} = \frac{\left|\Psi_{\xi x} - \Psi\right|}{\sum_{t=1}^{M} \Psi_{\Omega t}} \tag{9.6}$$

where,

$$\Psi = \sum_{\Omega t \neq \Omega x} \frac{\Psi_{\Omega t}}{(M-1)}$$

Find class that has the maximum value. If two or more classes take the maximum value then the classification of a pattern is rejected.

9.4 Fuzzy Classifier System

- **Generation of an Initial Population**

An initial population (the number of fuzzy if-then rules) of fuzzy if-then rules are generated by the rule generation procedure. Antecedent fuzzy sets of each fuzzy if then rule are randomly selected from the fuzzy sets.

- **Learning of Fuzzy if-then rules**

Each of the given training patterns is classified by the fuzzy if-then rules in the current population and that a training pattern is classified by the fuzzy if-then rule that satisfies the following relation.

$$\mu^{P}_{\hat{i}1}(\xi_{1p}).\mu^{P}_{\hat{i}2}(\xi_{1p}). \ldots \mu^{P}_{\hat{i}n}(\xi_{np}).GC^{P}_{\hat{i}j} = \text{Max}\,[\mu^{P}_{i1}(\xi_{1p}).\mu^{P}_{i2}(\xi_{1p}). \ldots \mu^{P}_{in}(\xi_{np}).GC^{P}_{ij} \tag{9.7}$$

When a training pattern is correctly classified by the fuzzy if-then rule the grade of certainty of that rule is increased as follows.

$$GC_{ij}^{P} = GC_{ij}^{P} + \eta_1(1 - GC_{ij}^{P}) \tag{9.8}$$

where η_1 is a positive learning rate for increasing the grade of certainty. On the other hand when a pattern is misclassified by the fuzzy if-then rule the grade of certainty of that rule is decreased as follows.

$$GC_{ij}^{P} = GC_{ij}^{P} - \eta_2 . GC_{ij}^{P} \tag{9.9}$$

where η_2 is a positive learning rate for decreasing the grade of certainty. This learning method is iterated in each generation.

- **Fitness Evaluation**

The fitness value of each fuzzy if-then rule is defined by

$$F(R_j) = W_{NCP}.NCP(R_j) - W_{NMP}(R_j) \tag{9.10}$$

where $N_{CP}(RJ)$ is the number of correctly classified training patterns by rule Rj, $N_{MP}(Rj)$ is the number of misclassified training patterns by Rj, and W_{NCP}+ W_{NMP} are positive weights. The fitness value of each rule is calculated in each generation.

- **Selection**

A pair of fuzzy if-then rules is selected from the current population to generate new fuzzy if-then rules for the next population. Each fuzzy if-then rule in the current populations is selected by the following selection probability.

$$P(R_j) = \frac{\text{fitness}(R_j) - \text{fitness}_{\min}(S)}{\sum_{R_k \in S} \{\text{fitness}(R_k) - \text{fitness}_{\min}(S)\}} \tag{9.11}$$

- **Crossover**

From each of the selected pairs of fuzzy if-then rules, the uniform crossover for the antecedent fuzzy sets generates two fuzzy if-then rules. The consequent class and the grade of certainty of each of the generated fuzzy if-then rules are determined by the rule generation procedure. That is only the antecedent fuzzy sets are mated.

- **Mutation**

Each antecedent fuzzy set of the generated fuzzy if-then rules by the crossover operation is randomly replaced with a different fuzzy set with the mutation probability. The consequent class and the grade of certainty of the mutated fuzzy if-then rule are determined by the rule generation procedure.

- **Replacement**

A certain proportion of fuzzy if-then rules in the current population is replaced with new fuzzy if-then rules generated by the crossover and mutation operations.

- **Termination test**

The total number of generations for terminating the execution of fuzzy classifier system is used.

9.5 Simulation Results

The training set consists of 43 patterns of class 1 and 57 patterns of class 2. The testing set has 47 patterns of class 1 and 53 patterns in class 2. The output class is either healthy (class 0) or with heart disease (class 1). GA parameters are shown in Table 9-1. Simulation results are presented in Table 9-2.

Table 9-1: GA parameters.

Population size	100
Replacement proportion	0.2
Mutation probability	0.1
Learning Rates	$\eta_1 = 0.001$ $\eta_2 = 0.1$
Number of iterations	100
Weights	$W_{NCP} = 1$ $W_{NMC} = 5$
Stopping Condition	100

Table 9-2: Using bell-shaped membership function

	# of patterns	Classification	error	Reject rate
# Training Patterns	100	90%	10%	0.0%
# Testing Patterns	100	89%	11%	0.0%

9.6 Conclusion

In this chapter the applicability of fuzzy classifier system is demonstrated by application to real-world pattern classification problem of Coronary Artery Disease. It is shown that this classifier could find a set of fuzzy if then rules with higher classification performance. Each fuzzy if-then rule was treated as an individual. The fitness value of each fuzzy if-then was determined by the numbers of correctly and wrongly classified training patterns by that rule.

One of the most significant advantages of these fuzzy rule based systems is their comprehensibility that is human users can easily understand these fuzzy if then rule because they are generally expressed as linguistic values. Fuzzy rule based classification methods, which generate fuzzy rules from numerical data, can be viewed as a knowledge acquisition tool for classification problems.

Chapter 10

General Conclusions and Further Directions of Research

10.1 Conclusions

A primary objective of this study is to assess the performance of computer programs in diagnosing CAD on the basis of clinical data. The diagnosis of CAD is complex decision making process. Therefore there is an obvious need for a clinical decision aid for accurate diagnosis based on enormous amount of existing knowledge. In cases where the difference between normal and abnormal is not clear, such as in the early detection of disease we have to deal with ambiguous information. Medical diagnosis involves making judgements about a patient's illness using specialist knowledge. The observations of symptoms includes results of tests, direct observation of the main complaint, various signs from the patient himself, and the patient's medical history. The results of this study demonstrates how computer programs can assist clinicians in the medical decision of diagnosing CAD. For a computer-based diagnostic aid to be of value it must be statistically valid, diagnostically accurate and its use must enhance the clinician's diagnostic performance. As with all clinical decision aids though, the true performance of the system can only be assessed by a formal clinical trial.

A genetic-programming-based feature classifier is proposed. Genetic-based machine learning systems use genetic search as their primary discovery heuristic. The most common genetic-based machine learning architecture is called classifier system which learns syntactically simple string IF-THEN rules so called classifiers to guide its performance in an arbitrary environment. The combined use of fuzzy logic and genetic algorithms and neural networks and genetic algorithms is also demonstrated.

The reader is provided with an introduction to pattern recognition in and a comprehensive overview of supervised classification.

Chapter 3 presentes an overview of the perceptron and MLP. In this chapter, learning techniques, limitations of perceptron, mathematical theory of MLP with practical issues are presented.

Chapter 4 focuses on the genetic algorithms and their mathematical foundations.

Chapter 5 deals with basic concepts of fuzzy set theory. Fuzzy sets are introduced to establish a mathematical framework to deal with problems in which the source of imprecision is the absence of sharply defined criteria for defining class membership.

Chapter 6 presents simulation results of neural networks. In this chapter an attempt has been made to formulate the neural network training criteria in medical diagnosis of CAD. Also, results are compared with various neural networks such as modular networks, radial basis function, reinforcement learning and backpropagation.

Chapter 7 describes the experimental results of hybrid system. In this chapter, the training of multiple layer perceptron by the GA optimization method is presented. Also, it is compared with the backpropagation algorithm. Experimental results are presented using 1-point crossover, 2-point crossover, uniform crossover, NN-specific 2-point crossover, and NN-specific uniform crossover.

Chapter 8 presents the experimental results of genetic programming. The Evolutionary Pre-Processor has been applied and compared with seven other classification methods.

Chapter 9 presents the experimental results of fuźzy classifier system combined with genetic algorithms. Here each fuzzy if-then rule was treated as an individual. The fitness value of each fuzzy if-then was determined by the numbers of correctly and wrongly classified training patterns by that rule.

All experiments are carried on the diagnosis of coronary artery disease, a real-world pattern classification problem.

The application of fuzzy sets to the field of artificial intelligence has given rise to a considerable number of possible techniques of interest in many disciplines. Fuzzy numbers, fuzzy algebra, fuggy logic inference, and fuzzy relations are well establilshed and have been used extensively in various fields of industry. A survey of methodologies into which fuzzy concepts have been incorporated includes fuzzy regression models, statistical decision making using fuzzy probablity and fuzzy entropy, fuzzy quantification theory, fuzzy mathematical programming, evaluation using fuzzy measures, diagnosis using fuzzy relations, fuzzy control and inference, multistage decision making using fuzzy dynamics programming, fuzzy databases and information retrieval using fuzzyfication functions and fuzzy expert systems. Conventional methods are good for simpler problems, while fuzzy systems are suitable for complex problems or applications that involve human descriptive or intuitive thinking. Fuzzy logic is also useful in understanding and simplifying the processing when the system behavior requires a complicated mathematical model. There are no generally accepted strict laws expressed in precise mathematics form as in hard disciplines such as physics. This kind of soft discipline provides ideal areas of application of fuzzy methods. Hence to enable a system to tackle real-life situations in a manner more like humans. The essential part of fuzzy system is fuzzy sets and fuzzy logic.

Fuzzy systems based on fuzzy if-then rules are successfully applied to various control problems such as truck backing, broom balancing and so forth. Fuzzy systems approaches also allow us to represent descriptive or qualitative expressions such as "definite" or "probable" and these are easily incorporated with symbolic statements. These expressions and representations are more natural than mathematical equations for many human judgement rules and statements. Fuzzy systems base their decisions on inputs in the form of linguistic variables. The variables are tested with a small number of IF-Then rules, which produce one or more responses depending on which rules were asserted. The response of each rule is weighted according to the confidence or degree of membership of its inputs. If the knowledge about the system behavior is described in approximate form or heuristic rules, then fuzzy system is suitable. Fuzzy systems lack capabilities of learning and no memory. This is why hybrid systems, particularly, GA/fuzzy systems, are becoming popular for certain applications. The fundamental concept of such hybrid systems is to complement each others weaknesses, thus creating new approaches to solve problems. One of the most significant differences between control problems and classification problems is the

dimensionality of inputs. While control problems usually involve only a few inputs, real-world pattern classification problems often involve several attributes.

10.2 Implications for Further Research

In the context of fuzzy system design uncertainties are represented by membership functions design and by their individual properties. The geometrical shape of a membership function is the characterization of uncertainty in the corresponding fuzzy variable. The triangle and the trapezoid are the two geometric shapes commonly used to represent uncertainties. Therefore a high level of detail in shape design must be considered as a conceptual error.

Determining or tuning good membership functions and fuzzy rules is not always easy. Even after extensive testing, it is difficult to say how many membership functions are really required. The shape of the membership function can not be formed arbitrarily because arbitrary design can produce unpredictable results in the basic fuzzy inference algorithm. The design challenge is to employ a reasonable level of detail when forming membership functions so that the basic fuzzy inference algorithm behaves as expected. The shape effects will be examined in more detail useful for design.

Genetic algorithms are particularly well suited for tuning the membership functions in terms of placing them in the universe of discourse. GAs can be used to compute membership functions. Given some functional mapping for a system, same membership functions and their shapes are assumed for the various fuzzy variables defined for a problem. These membership functions are than coded as bit strings that are then concatenated. An evaluation (fitness) function is used to evaluate the fitness of each set of membership functions (parameter that define the functional mapping). Properly configured genetic algorithm/fuzzy architectures search the complete universe of discourse and find adequate solutions according to the fitness function.

Appendix A

Generalised Delta Learning Rule for MLP

Learning is a relatively permanent change in behavior brought about by experience. In artificial neural networks the following general learning rule is adopted: The weight vector increases in proportion to the product of input and learning signal. The back-propagation algorithm allows to extract input/output mapping knowledge within multilayer networks. In the course of learning the synaptic weights as well as the thresholds are adjusted so that the current least mean-square error is reduced until all mapping examples from the training set are learned within an acceptable overall error. The delta learning rule is only valid for continuous activation functions and in the supervised training mode.

The gradient descent formula is given by

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \qquad \text{for } j \text{ and } i = 1, 2, \ldots, I \tag{A.1}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial (net_j)} \cdot \frac{\partial (net_j)}{\partial w_{ji}} \tag{A.2}$$

Therefore we may express the weight adjustments as

$$\Delta w_{ji} = \eta \delta_{yj} x_i \tag{A.3}$$

where δ_{yj} is the error signal term of the hidden layer having output *y*. This error signal term is produced by the j'th neuron of the hidden layer, where j = 1, 2, ..., J. The error signal term is equal to

$$\delta_{yj} = -\frac{\partial E}{\partial (net_j)} \quad \text{for } j = 1, 2, \ldots, J \tag{A.4}$$

In contrast to the output layer neurons' excitation net_k which affected the k'th neuron output only, the net_j contributes now to every error component in the error sum containing K terms in $\frac{1}{2}\sum_{k=1}^{K}\left(d_{pk} - o_{pk}\right)^2$ for k = 1, 2, ..., K , for a specific pattern p. The error signal term δ_{yj} at the node j can be computed as follows:

$$\delta_{yj} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (net_j)} \tag{A.5}$$

where

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{K}(d_k - f(net_k(y))^2\right) \tag{A.6}$$

and the second term of (5) is equal to

$$\frac{\partial y_j}{\partial (net_j)} = f'(net_j) \tag{A.7}$$

Calculations of (A.6) result in

$$\frac{\partial E}{\partial y_j} = -(d_k - o_k)\frac{\partial}{\partial y_j}\left\{f\left[net_k(y)\right]\right\} \qquad (A.8)$$

Calculation of the derivative of equation (8) yields

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}(d_k - o_k)\, f'(net_k)\frac{\partial(net_k)}{\partial y_j} \qquad (A.9)$$

This equation can be simplified as

$$\frac{\partial E}{\partial y_j} = \Sigma \delta_{ok} w_{kj} \qquad (A.10)$$

Combining (7) and (10) results in rearranging ∂y_j expressed in (5) to the form

$$\partial_{yj} = f'(net_j)\ \Sigma \delta_{ok} w_{kj} \text{ for } j = 1, 2, \ldots, J \qquad (A.11)$$

The weight adjustment (3) in the hidden layer now becomes

$$\Delta w_{ji} = \eta f'(net_j) x_i \sum_{k=1}^{K} \delta_{ok} w_{kj} \text{ for } j = 1, 2, \ldots, J \text{ and } i = 1, 2, \ldots, I \qquad (A.12)$$

This equation is known as *generalized delta learning rule.*

Appendix B

Learning of the weights of MLP using GA

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "sugal.h"

#define NREPOCHS 1
#define MAXLAYERS 5
#define MAXTRSETS 500
#define TOLERANCE 0.4
#define CONFIGFILE "nnlearn5.cfg"
#define OUTPUTFILE "nnlearn5.out"

int Evaluate(SuChromosome *chrom, double *fitness);
void PrintBestChromosome();
int Initialisation(void);
double CalcError(SuChromosome *chrom);
void GetWeightsFromChromosome(SuChromosome *chrom);
void GetWeightsFromString(unsigned char *string);
void SetWeightsToString(unsigned char *string);
void GetLastStepsFromString(unsigned char *string);
void SetLastStepsToString(unsigned char *string);
void CalcOutput(int TrSetIndex);
double Activation(double sum);
void InputData(void);
void AllocMemory(void);
void FreeMemory(void);
void ReadTrainingSet(void);
void MyExit(char *ErrMsg);
void MyCrossover1(SuChromosome *p1, SuChromosome *p2, SuChromosome *c1, SuChromosome
*c2);
void MyCrossover2(SuChromosome *p1, SuChromosome *p2, SuChromosome *c1, SuChromosome
*c2);
void BPMutation(unsigned char *string, int offset);
void BPMomentum(unsigned char *string, int offset);
void BPHybrid(unsigned char *string, int offset);
void SuWriteAugmentedChromosome( SuAugmentedChromosome *aug_chrom );

char NNFile[80],TRFile[80];
FILE *fp;
int LearningType;
```

```
int AllocMem = FALSE;

int NumLayers;
int NumCells[MAXLAYERS];
int NumWeights[MAXLAYERS];
int TotalNumWeights;
int XPoints[50];
int NrInputs, NrOutputs;
double *CellWeights[MAXLAYERS];
double *DeltaWeights[MAXLAYERS];
double *LastStep[MAXLAYERS];
double *CellOutputs[MAXLAYERS];
double TotWeights[20]; /* not fit for multi-layered networks!! */

int NumTrSets;
double **TrInputs;
double **TrOutputs;

void PrintBestChromosome()
{
        char msg[50];
        sprintf( msg, "Best : ");
        SuOutput( msg );
        SuWriteAugmentedChromosome( suThePool->best );
}

void MyCrossover1(SuChromosome *p1, SuChromosome *p2, SuChromosome *c1, SuChromosome
*c2)
/* Special crossover routine, gives uniform crossover, but only between cell-units
        (a cell-unit is the group of weights of all the inputs of one neuron) */
{
 int i, j, which_way;

 for ( i=1; i < (XPoints[0]+2); i++ )
 {
        which_way = SuRandBit();

        if (which_way)
                for (j= XPoints[i]; j<XPoints[i+1]; j++)
                {
                        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p1->string, j));
                        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p2->string, j));
                }
        else
                for (j= XPoints[i]; j<XPoints[i+1]; j++)
                {
                        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p2->string, j));
                        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p1->string, j));
                }
 }
}

void MyCrossover2(SuChromosome *p1, SuChromosome *p2, SuChromosome *c1, SuChromosome
*c2)
/* Special crossover routine, gives 2-point crossover, but only between cell-units
        (a cell-unit is the group of weights of all the inputs of one neuron) */
{
 int j;
 int x_point1 = 2 + SuRandIntUpTo(XPoints[0]); /* 2 <= x_point1 <= XPoints[0]+1 */
 int x_point2 = 2 + SuRandIntUpTo(XPoints[0]); /* ,,  x_point2     ,,      */
```

```
int temp;


/* Swap over the crossover points if not in the right order */
if ( x_point1 > x_point2 )
{
        temp = x_point1;
        x_point1 = x_point2;
        x_point2 = temp;
}

/* Copy starts */
for (j=XPoints[1]; j<XPoints[x_point1]; j++)
{
        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p1->string, j));
        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p2->string, j));
}

/* Copy middles */
for (j=XPoints[x_point1]; j<XPoints[x_point2]; j++)
{
        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p2->string, j));
        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p1->string, j));
}

/* Copy ends */
for (j=XPoints[x_point2]; j<XPoints[XPoints[0]+2]; j++)
{
        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p1->string, j));
        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p2->string, j));
}
}

void MyCrossover3(SuChromosome *p1, SuChromosome *p2, SuChromosome *c1, SuChromosome
*c2)
/* Special crossover 2 routine, gives uniform crossover, but only between cell-units
        (a cell-unit is the group of weights of all the inputs of one neuron)
        also sees the weight of its connection to the next cell as part of this cell-unit */
{
 int i, j, which_way, parentstart, parent;

 parentstart = XPoints[  (XPoints[0]+2-NrOutputs)  ];
 for ( i=1; i < (XPoints[0]+2-NrOutputs); i++ )
 {
        which_way = SuRandBit();

        if (which_way)
        {
                for (j= XPoints[i]; j<XPoints[i+1]; j++)
                {
                        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p1->string, j));
                        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p2->string, j));
                }
                for (j=0; j<NrOutputs; j++)
                {
                        parent = parentstart + i + (j * (1 + NumCells[1]));
                        SuSetNativeDouble(c1->string, parent, SuGetNativeDouble(p1->string,
parent));
                        SuSetNativeDouble(c2->string, parent, SuGetNativeDouble(p2->string,
parent));
```

```
                }
        }
        else
        {
                for (j= XPoints[i]; j<XPoints[i+1]; j++)
                {
                        SuSetNativeDouble(c1->string, j, SuGetNativeDouble(p2->string, j));
                        SuSetNativeDouble(c2->string, j, SuGetNativeDouble(p1->string, j));
                }
                for (j=0; j<NrOutputs; j++)
                {
                        parent = parentstart + i + (j * (1 + NumCells[1]));
                        SuSetNativeDouble(c1->string, parent, SuGetNativeDouble(p2->string,
parent));
                        SuSetNativeDouble(c2->string, parent, SuGetNativeDouble(p1->string,
parent));
                }
        }
  }
}

void BPHybrid( unsigned char *string, int offset )
{
        int i,j,k,l, place, WeightOffset;
        double Error, IncError, tempoutput, CellFactor;
        double eta, OldError;
        int h=0;
        char buf[30];

        if (LearningType == 0)
                eta = 0.1;
        else
        {
                if (LearningType == 1)
                        eta = 0.5;
                else
                        eta = 1.0;
        }

        GetWeightsFromString(string);
        Error = 10000.0; /* initial 'very high' value */

        /* learning algorithm */
        do /* learn for x epochs, until Error doesn't increase anymore */
        {
                /* initialise stuff for each epoch */
                OldError = Error;
                Error = 0.0;
                for (i=1;i<NumLayers;i++)
                        for (j=0;j<NumWeights[i];j++)
                                DeltaWeights[i][j] = 0.0;

                for (i=0;i<NumTrSets;i++) /* for every training vector */
                {
                        CalcOutput(i);
                        for (j=0;j<NumCells[2];j++) /* for every output neuron */
                        {
                                WeightOffset = j * (NumCells[1] + 1);
                                tempoutput = CellOutputs[2][j];
        /* = y3 */
```

```
                                IncError = TrOutputs[i][j] - tempoutput;                /* = t3 - y3 */
                                Error += (IncError * IncError);
                                CellFactor = IncError * (tempoutput * (1.0 - tempoutput));

                                /* calculate weight changes of the output neuron */
                                DeltaWeights[2][WeightOffset] += CellFactor;
                                for (k=0;k<NumCells[1];k++) /* for every weight of this output
neuron */
                                        DeltaWeights[2][WeightOffset+k+1] += CellFactor *
CellOutputs[1][k];

                                /* calculate weight changes of the hidden neurons */
                                place = 0;
                                for (k=0;k<NumCells[1];k++) /* for every hidden neuron */
                                {
                                        tempoutput = CellOutputs[1][k] * (1.0 - CellOutputs[1][k]);
/* = y1(1-y1) */
                                        DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput;
                                        for (l=0;l<NumCells[0];l++) /* for every weight of this
hidden neuron */
                                                DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput * CellOutputs[0][l];
                                }
                        }
                }

                /* update cell weights after each learning step */
                if (Error < OldError)
                {
                        h++;
                        for (i=1;i<NumLayers;i++)
                                for (j=0;j<NumWeights[i];j++)
                                        CellWeights[i][j] += eta * DeltaWeights[i][j];
                }
        } while ((Error < OldError) && (h<100));
        sprintf(buf,"%d, ",h);
        SuOutput(buf);

        SetWeightsToString(string);
}

void BPMutation( unsigned char *string, int offset )
{
        int h,i,j,k,l, place, WeightOffset;
        double Error, IncError, tempoutput, CellFactor;
        double eta;

        if (LearningType == 0)
                eta = 5.0 / NumTrSets;
/*              eta = 2.0;
        else
        {
                if (LearningType == 1)
                        eta = 0.5;
                else
                        eta = 1.0;
        }
*/ else
                eta = LearningType / 100.0;
```

```
	GetWeightsFromString(string);

	/* learning algorithm */
	for (h=0;h<NREPOCHS;h++) /* learn for NREPOCHS epochs */
	{
		/* initialise stuff for each epoch */
		Error = 0.0;
		for (i=1;i<NumLayers;i++)
			for (j=0;j<NumWeights[i];j++)
				DeltaWeights[i][j] = 0.0;

		for (i=0;i<NumTrSets;i++) /* for every training vector */
		{
			CalcOutput(i);
			for (j=0;j<NumCells[2];j++) /* for every output neuron */
			{
				WeightOffset = j * (NumCells[1] + 1);
				tempoutput = CellOutputs[2][j];
	/* = y3 */
				IncError = TrOutputs[i][j] - tempoutput;		/* = t3 - y3 */
				Error += (IncError * IncError);
				CellFactor = IncError * (tempoutput * (1.0 - tempoutput));

				/* calculate weight changes of the output neuron */
				DeltaWeights[2][WeightOffset] += CellFactor;
				for (k=0;k<NumCells[1];k++) /* for every weight of this output
neuron */
					DeltaWeights[2][WeightOffset+k+1] += CellFactor *
CellOutputs[1][k];

				/* calculate weight changes of the hidden neurons */
				place = 0;
				for (k=0;k<NumCells[1];k++) /* for every hidden neuron */
				{
					tempoutput = CellOutputs[1][k] * (1.0 - CellOutputs[1][k]);
/* = y1(1-y1) */
					DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput;
					for (l=0;l<NumCells[0];l++) /* for every weight of this
hidden neuron */
						DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput * CellOutputs[0][l];
				}
			}
		}

		/* update cell weights after each learning step */
		for (i=1;i<NumLayers;i++)
			for (j=0;j<NumWeights[i];j++)
				CellWeights[i][j] += eta * DeltaWeights[i][j];
	}

	SetWeightsToString(string);
}

void BPMomentum(unsigned char *string, int offset)
{
	int h,i,j,k,l, place, WeightOffset;
	double Error, IncError, tempoutput, CellFactor, next_step;
	double eta, momentum;
```

```
if (LearningType == 0)
{       eta = 0.5; momentum = 0.5; }
else
{
        if (LearningType == 1)
        {       eta = 0.1; momentum = 0.9; }
        else
        {       eta = 0.3; momentum = 0.5; }
}

GetWeightsFromString(string);
GetLastStepsFromString(string);

/* learning algorithm */
for (h=0;h<NREPOCHS;h++) /* learn for NREPOCHS epochs */
{
        /* initialise helpstuff */
        Error = 0.0;
        for (i=1;i<NumLayers;i++)
                for (j=0;j<NumWeights[i];j++)
                        DeltaWeights[i][j] = 0.0;

        for (i=0;i<NumTrSets;i++) /* for every training vector */
        {
                CalcOutput(i);
                for (j=0;j<NumCells[2];j++) /* for every output neuron */
                {
                        WeightOffset = j * (NumCells[1] + 1);
                        tempoutput = CellOutputs[2][j];
/* = y3 */
                        IncError = TrOutputs[i][j] - tempoutput;        /* = t3 - y3 */
                        Error += (IncError * IncError);
                        CellFactor = IncError * (tempoutput * (1.0 - tempoutput));

                        /* calculate weight changes of the output neuron */
                        DeltaWeights[2][WeightOffset] += CellFactor;
                        for (k=0;k<NumCells[1];k++) /* for every weight of this output
neuron */
                                DeltaWeights[2][WeightOffset+k+1] += CellFactor *
CellOutputs[1][k];

                        /* calculate weight changes of the hidden neurons */
                        place = 0;
                        for (k=0;k<NumCells[1];k++) /* for every hidden neuron */
                        {
                                tempoutput = CellOutputs[1][k] * (1.0 - CellOutputs[1][k]);
/* = y1(1-y1) */
                                DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput;
                                for (l=0;l<NumCells[0];l++) /* for every weight of this
hidden neuron */
                                        DeltaWeights[1][place++] += CellFactor *
CellWeights[2][WeightOffset + k + 1] * tempoutput * CellOutputs[0][l];
                        }
                }
        }

        /* after each epoch: update cell weights */
        for (i=1;i<NumLayers;i++)
```

```
                {
                        for (j=0;j<NumWeights[i];j++)
                        {
                                next_step = (eta * DeltaWeights[i][j]) + (momentum * LastStep[i][j]);
                                LastStep[i][j] = next_step;
                                CellWeights[i][j] += next_step;
                        }
                }

        }

        SetWeightsToString(string);
        SetLastStepsToString(string);
}


int Evaluate(SuChromosome *chrom, double *fitness)
/* Evaluates chromosome (assigns it a fitness value) */
{
        suNNTotalRuns++;
        *fitness = CalcError(chrom);
        return 0;
}

double CalcError(SuChromosome *chrom)
/* Calculates total network error on the training set */
{
        int i,j,NumIncorrect;
        double Error = 0.0, IncError;
        char string[100];

        GetWeightsFromChromosome(chrom);

        /* compare calculations with TrOutputs */
        NumIncorrect = 0;
        for (i=0;i<NumTrSets;i++)
        {
                CalcOutput(i);
                for (j=0;j<NrOutputs;j++)
                {
                        IncError = CellOutputs[NumLayers-1][j] - TrOutputs[i][j];
                        Error += (IncError * IncError);
                        NumIncorrect += (fabs(CellOutputs[NumLayers-1][j] -

                                        TrOutputs[i][j]) >= TOLERANCE);
                }
        }

        if ((NumIncorrect==0) && (suNNSolutionFound==0) && (suNNStopWhenLearned==1))
        {
                sprintf(string,
                 "1st perfect network found after %d evaluations (Gen %d, Err = %lf)\n",
                 suNNTotalRuns,suGeneration,Error);
                SuOutput(string);
                suNNSolutionFound = 1;
        }
        if (Error < suNNBestError)
        {
                /* remember the error of the best network of this generation */
                suNNBestError = Error;
```

```
                suNNBestPercCorrect = 100.0 - (100.0 * (float) NumIncorrect) / (float)
(NrOutputs*NumTrSets);
        }

        return Error;
}

void CalcOutput(int TrSetIndex)
/* Calculates the network output on one training sample */
{
        int i,j,k;
        double sum;
        int index;

        /* 1st layer: output = input */
        for (i=0;i<NrInputs;i++)
                CellOutputs[0][i] = TrInputs[TrSetIndex][i];

        /* other layers: output = sigmoid(weights x inputs)  */
        for (i=1;i<NumLayers;i++)
        {
                index = 0;
                for (j=0;j<NumCells[i];j++)
                {
                        sum = CellWeights[i][index++];
                        for (k=0;k<NumCells[i-1];k++)
                                sum += CellWeights[i][index++] * CellOutputs[i-1][k];
                        CellOutputs[i][j] = Activation(sum);
                }
        }
}

double Activation(double sum)
/* Calculate sigmoid transfer function of the neurons*/
{
        /* prevent overflow */
        if (sum < -30.0) return 0.0;
        if (sum >  30.0) return 1.0;
        return (1.0/(1.0+exp(-sum)));
}

void GetWeightsFromChromosome(SuChromosome *chrom)
/* Extracts weights from chromosome and stores them in CellWeights[][]
        Optionally regroups the cells according to 'importance' */
{
        int i,j,k=0;
        int index1,index2;
        double dummy;
        static int SortType=0;

        for (i=1;i<NumLayers;i++)
                for (j=0;j<NumWeights[i];j++)
                        CellWeights[i][j] = SuGetNativeDouble(chrom->string,k++);

        /* If required, sort cells according to 'importance' */
        /* not fit for multi-layered networks yet!! */
        if(suNNRegroupCells)
        {
                /* Clear old TotWeights[] (='importance') */
                for (j=0;j<NumCells[1];j++)
```

```
                TotWeights[j] = 0.0;

        /* Calculate new TotWeights[] */
        for (i=0;i<NrOutputs;i++)
        {
                k = (NumCells[1] + 1) * i + 1;
                for (j=0;j<NumCells[1];j++)
                        TotWeights[j] += fabs(CellWeights[2][k+j]);
        }

        /* sort cells according to importance */
        if ((suNNRegroupCells == 1) || ((suNNRegroupCells == 2) && (SortType == 0)))
        {
                /* sort in descending order */
                for (i=0;i<NumCells[1]-1;i++)
                        for (j=i+1;j<NumCells[1];j++)
                                if(TotWeights[i] < TotWeights[j])
                                {
                                        /* exchange cells */
                                        dummy = TotWeights[i];
                                        TotWeights[i] = TotWeights[j];
                                        TotWeights[j] = dummy;

                                        index1 = (NrInputs + 1) * i;
                                        index2 = (NrInputs + 1) * j;
                                        for (k=0;k<=NrInputs;k++)
                                        {
                                                dummy = CellWeights[1][index1 + k];
                                                CellWeights[1][index1 + k] =
CellWeights[1][index2 + k];
                                                CellWeights[1][index2 + k] = dummy;
                                        }
                                        for (k=0;k<NrOutputs;k++)
                                        {
                                                index1 = (NumCells[1] + 1) * k + 1;
                                                dummy = CellWeights[2][index1 + i];
                                                CellWeights[2][index1 + i] =
CellWeights[2][index1 + j];
                                                CellWeights[2][index1 + j] = dummy;
                                        }
                                }
                                else {} /* do nothing */
                SortType = 1;
        }
        else if ((suNNRegroupCells == 2) && (SortType == 1))
        {
                /* sort in ascending order */
                for (i=0;i<NumCells[1]-1;i++)
                        for (j=i+1;j<NumCells[1];j++)
                                if(TotWeights[i] > TotWeights[j])
                                {
                                        /* exchange cells */
                                        dummy = TotWeights[i];
                                        TotWeights[i] = TotWeights[j];
                                        TotWeights[j] = dummy;

                                        index1 = (NrInputs + 1) * i;
                                        index2 = (NrInputs + 1) * j;
                                        for (k=0;k<=NrInputs;k++)
                                        {
```

```
                                                dummy = CellWeights[1][index1 + k];
                                                CellWeights[1][index1 + k] =
CellWeights[1][index2 + k];
                                                CellWeights[1][index2 + k] = dummy;
                                        }
                                        for (k=0;k<NrOutputs;k++)
                                        {
                                                index1 = (NumCells[1] + 1) * k + 1;
                                                dummy = CellWeights[2][index1 + i];
                                                CellWeights[2][index1 + i] =
CellWeights[2][index1 + j];
                                                CellWeights[2][index1 + j] = dummy;
                                        }
                                }
                                else {} /* do nothing */
                        SortType = 0;
                }

                /* Store sorted setup back in chromosome */
                k = 0;
                for (i=1;i<NumLayers;i++)
                        for (j=0;j<NumWeights[i];j++)
                                SuSetNativeDouble(chrom->string,k++,CellWeights[i][j]);
        }
}

void GetWeightsFromString( unsigned char *string)
{
        int i,j,k=0;

        for (i=1;i<NumLayers;i++)
                for (j=0;j<NumWeights[i];j++)
                        CellWeights[i][j] = SuGetNativeDouble(string,k++);
}

void GetLastStepsFromString( unsigned char *string)
{
        int i,j,k=TotalNumWeights;

        if (suGeneration==1)
                for (i=1;i<NumLayers;i++)
                        for (j=0;j<NumWeights[i];j++)
                                LastStep[i][j] = 0.0;
        else
                for (i=1;i<NumLayers;i++)
                        for (j=0;j<NumWeights[i];j++)
                                LastStep[i][j] = SuGetNativeDouble(string,k++);
}

void SetWeightsToString( unsigned char *string)
{
        int i,j,k=0;

        for (i=1;i<NumLayers;i++)
                for (j=0;j<NumWeights[i];j++)
                        SuSetNativeDouble(string,k++,CellWeights[i][j]);
}

void SetLastStepsToString( unsigned char *string)
{
```

```
        int i,j,k=TotalNumWeights;

        for (i=1;i<NumLayers;i++)
                for (j=0;j<NumWeights[i];j++)
                        SuSetNativeDouble(string,k++,LastStep[i][j]);
}

int Initialisation(void)
{
        static int Initialised = FALSE;

        /* only first time: initialisation */
        if (!Initialised)
        {
                InputData();
                AllocMemory();
                ReadTrainingSet();
                Initialised = TRUE;
                sprintf(suNNString[0],"Parameters: ");
        }
        return(0);
}

void InputData()
/* Sets neural network configuration */
{
        FILE *InputFile;
        int i,j,ch,index;
        char text[80];

        if ((InputFile = fopen(CONFIGFILE,"r")) == NULL)
                MyExit("Missing configuration file 'nnlearn.cfg'");
        /* read name of neural network (.nn) file */
        do ch = getc(InputFile); while (ch!='#');
        fgets(NNFile,80,InputFile);
        NNFile[strlen(NNFile)-1]=0;
        /* read name of training set (.tr) file */
        do ch = getc(InputFile); while (ch!='#');
        fgets(TRFile,80,InputFile);
        TRFile[strlen(TRFile)-1]=0;
        fclose(InputFile);

        /* read neural network configuration from specified '.nn' file */
        if ((InputFile = fopen(NNFile,"r")) == NULL)
                MyExit("Missing input '.nn' file");

        /* Learning Type
                BPMutation:          BPMomentum:
                0: eta = 0.1, mu = 0                0: eta = 0.5, mu = 0.5
                1: eta = 0.5, mu = 0   1: eta = 0.1, mu = 0.9
                2: eta = 1.0, mu = 0         2: eta = 0.3, mu = 0.5  */
        do ch = getc(InputFile); while (ch!=':');
        if (fscanf(InputFile,"%d",&LearningType) == EOF)
                MyExit("Premature end of '.nn' file (LearningType)");

        /* StopWhenLearned
                0: don't stop SUGAL
                1: stop SUGAL when a network has been found that classifies all training
                        data correctly (only works when SUGAL is in single run mode) */
        do ch = getc(InputFile); while (ch!=':');
```

```
        if (fscanf(InputFile,"%d",&suNNStopWhenLearned) == EOF)
                MyExit("Premature end of '.nn' file (StopWhenLearned)");

        /* ContinueXRuns (only in combination with StopWhenLearned)
                continues SUGAL for ContinueXRuns generations after a perfect classifying
                network has been found (has no effect when StopWhenLearned=0 and/or when
                SUGAL is in multiple run mode) */
        do ch = getc(InputFile); while (ch!=':');
        if (fscanf(InputFile,"%d",&suNNContinue) == EOF)
                MyExit("Premature end of '.nn' file (ContinueXRuns)");

        /* TextOutput: Affects textoutput to 'nnlearn5.out' file only
                0: write nothing to file
                1: write everything to file */
        do ch = getc(InputFile); while (ch!=':');
        if (fscanf(InputFile,"%d",&suNNTextOutput) == EOF)
                MyExit("Premature end of '.nn' file (TextOutput)");
        suNNPrintBest = suNNTextOutput;

        /* RegroupCells:
                0: Don't regroup cells during evaluation
                1:      Sort cells according to descending 'importance' ('special' crossover should be
used)
                2: Same as 1, but sorted in both ascending & descending orders */
        do ch = getc(InputFile); while (ch!=':');
        if (fscanf(InputFile,"%d",&suNNRegroupCells) == EOF)
                MyExit("Premature end of '.nn' file (RegroupCells)");
/*      if (!SpecialCrossover) suNNRegroupCells = 0; */

        /* nr. of layers, followed by the nr. of cells in each layer
                (includes input and output layers) */
        do ch = getc(InputFile); while (ch!=':');
        if (fscanf(InputFile,"%d",&NumLayers) == EOF)
                MyExit("Premature end of '.nn' file (NumLayers)");
        if (NumLayers > MAXLAYERS) MyExit("Too many layers");
        for (i=0; i<NumLayers; i++)
        {
                do ch = getc(InputFile); while (ch!=':');
                if (fscanf(InputFile,"%d",&NumCells[i]) == EOF)
                        MyExit("Premature end of '.nn' file (NumCells)");
        }

        /* calculate the nr. of weights and the place of the MyCrossover points */
        TotalNumWeights=0;
        XPoints[0]=-1;
        XPoints[1]=0; /* start of first section */
        index=1;
        for (i=1; i<NumLayers; i++)
        {
                NumWeights[i] = (NumCells[i-1]+1)*NumCells[i];
                TotalNumWeights += NumWeights[i];
                for (j=0; j<NumCells[i]; j++)
                {
                        XPoints[index+1] = XPoints[index] + NumCells[i-1] + 1;
                        index++;
                }
                XPoints[0] += NumCells[i];
        }

        fclose(InputFile);
```

```
        NrInputs = NumCells[0];
        NrOutputs = NumCells[NumLayers-1];

        if ((suNNOutputFile = fopen( OUTPUTFILE,"w"))==NULL)
        {
                sprintf( text, "Failed to open output_file [%s]", OUTPUTFILE );
                SuWarning( text );
        }
}

void AllocMemory()
/* Allocates memory for neural network calculations */
{
        int i;

        /* Allocate Memory for cell weights + changes */
        for (i=1;i<NumLayers;i++)
        {
                if ((CellWeights[i] = malloc(NumWeights[i] * sizeof(double)))==NULL)
                        MyExit("Not enough memory (1)");
                if ((DeltaWeights[i] = malloc(NumWeights[i] * sizeof(double)))==NULL)
                        MyExit("Not enough memory (1)");
                if ((LastStep[i] = malloc(NumWeights[i] * sizeof(double)))==NULL)
                        MyExit("Not enough memory (1)");
        }

        /* Allocate Memory for cell outputs */
        for (i=0;i<NumLayers;i++)
                if ((CellOutputs[i] = malloc(NumCells[i] * sizeof(double)))==NULL)
                        MyExit("Not enough memory (1)");

        /* Allocate Memory for training sets */
        if ((TrInputs  = malloc(MAXTRSETS * sizeof(double *)))==NULL)
                MyExit("Not enough memory (2)");
        if ((TrOutputs = malloc(MAXTRSETS * sizeof(double *)))==NULL)
                MyExit("Not enough memory (2)");
        for (i=0;i<MAXTRSETS;i++)
        {
                if ((TrInputs[i]  = malloc(NrInputs  * sizeof(double)))==NULL)
                        MyExit("Not enough memory (3) ");
                if ((TrOutputs[i] = malloc(NrOutputs * sizeof(double)))==NULL)
                        MyExit("Not enough memory (3) ");
        }
        AllocMem = TRUE;
}

void FreeMemory()
/* Frees memory allocated by AllocMemory() */
{
        int i;

        if (AllocMem)
        {
                for (i=1;i<NumLayers;i++)
                {
                        free(CellWeights[i]);
                        free(DeltaWeights[i]);
                        free(LastStep[i]);
                }
                for (i=0;i<NumLayers;i++)
```

```
                free(CellOutputs[i]);
            for (i=0;i<MAXTRSETS;i++)
            {
                free(TrInputs[i]);
                free(TrOutputs[i]);
            }
            free(TrInputs);
            free(TrOutputs);
            if (suNNOutputFile)
            {
                fprintf(suNNOutputFile, "\nMemory freed, closing file...\n");
                fflush(suNNOutputFile);
                fclose(suNNOutputFile);
            }
            AllocMem = FALSE;
        }
}

void ReadTrainingSet()
/* Reads the training-samples from the '.tr' file and stores them in
        the arrays TrInputs[][] and TrOutputs[][] */
{
        int i,trsetnr;
        FILE *InputFile;

        if ((InputFile = fopen(TRFile,"r")) == NULL)
                MyExit("Missing input file nnlearn.tr");
        trsetnr = 0;
        while ((trsetnr<MAXTRSETS) &&
                        (fscanf(InputFile,"%lf",&(TrInputs[trsetnr][0])) != EOF))
        {
                for (i=1; i<NrInputs; i++)
                        if (fscanf(InputFile,"%lf",&(TrInputs[trsetnr][i])) == EOF)
                                MyExit("Premature end of trainingset file");
                for (i=0; i<NrOutputs; i++)
                        if (fscanf(InputFile,"%lf",&(TrOutputs[trsetnr][i])) == EOF)
                                MyExit("Premature end of trainingset file");
                trsetnr++;
        }
        fclose(InputFile);
        NumTrSets = trsetnr;
}

void MyExit(char *ErrMsg)
/* Prints error message and exits programme */
{
        printf("\n%s\n",ErrMsg);
        exit(1);
}
```

Appendix C

Fuzzy Classifier System

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>


typedef struct{
 int part;
 int posi;
}CONDITION;

typedef struct{
 CONDITION *condition; /* antecedent part */
 int clss;          /* consequent part */
 double cf;         /* certainty factor */
 double fitness;    /* fitness        */
 int correct;       /* correctly classified patterns by this rule */
 int wrong;         /* wrongly classified patterns by this rule */
}CLASSIFIER;

typedef struct{
 double *x;    /* attibute value */
 int   clss;  /* class */
 int   group;
}SAMPLE;

typedef struct{
 int      N_pop;    /* population size */
 double   P_rep;    /* replacement proportion */
 double   P_mut;    /* mutation Probability */
 int      N_learn;  /* learning iteration */
 double   eta_1;    /* eta_1 and eta_2 are learning rates */
 double   eta_2;
 double   W_ncp;    /* the weights in the fitness function */
 double   W_nmp;
 int      N_gen;    /* number of generations */
 char     *d_f_name; /* data file name */
 int      seed;     /* random seed */
 char     shape;    /* the shape of membership function */
 char     *part;    /* the number of partitions of each axis */
 /* --the followings are related to sample(training) data-- */
 int      s_num;    /* number of sample */
 int      s_dim;    /* dimension */
 int      s_cls;    /* class */
}PARAMETER;
```

```
typedef struct{
 int correct;
 int error;
 int reject;
 int r_disjoin;       /* number of disjoin rules */
}PERFORMANCE;

typedef struct{
 int total;
 int disjoin;       /* number of disjoin rules */
}I_SET;               /* the information of rule set( classifiers ) */

/*-------------------------------------------------------------*/

void usage( void );
void read_parafile( PARAMETER* );
void **alloc_2( int, int, int, int );
void myread( FILE*, char* );
void *alloc_1( int, int );
void free_2( void**, int );
void read_data1( FILE*, PARAMETER* );
void read_data2( FILE*, PARAMETER*, SAMPLE* );
void modify_parameter( int, char**, PARAMETER* );
void decide_consequent( CLASSIFIER*, SAMPLE*, PARAMETER*, int );
double membership( CONDITION*, double, PARAMETER* );
PERFORMANCE t_evaluate( CLASSIFIER*, SAMPLE*, PARAMETER*, int );
PERFORMANCE evaluate( CLASSIFIER*, SAMPLE*, PARAMETER*, int );
void learn_cf( CLASSIFIER*, SAMPLE*, PARAMETER*, int );
I_SET how_many_rules( CLASSIFIER*, PARAMETER* );
void preserve_elite( CLASSIFIER*, CLASSIFIER*, PARAMETER* );
void g_o( CLASSIFIER*, PARAMETER* );
void selection( double*, int, int*, int* );
void crossover( CLASSIFIER*, CLASSIFIER*, CLASSIFIER*, CLASSIFIER*, int );
void mutation( CLASSIFIER*, PARAMETER* );
void replace( CLASSIFIER*, CLASSIFIER*, PARAMETER*, int );
int select_rep( int, int*, CLASSIFIER*, int );
void out_rule( CLASSIFIER*, CLASSIFIER*, int, PARAMETER* );

/*-------------------------------------------------------------*/



#include"cs.h"

int main( int argc, char **argv )
{
 CLASSIFIER *current, *elite, *eelite;
 SAMPLE    *smp;
 PARAMETER  para;
 PERFORMANCE *c_perf, *e_perf, *ee_perf;
 PERFORMANCE *tc_perf, *te_perf, *tee_perf;
 I_SET      c_rule, e_rule, ee_rule;

 char buf[30], tmp_c;
 int i, j, n_gen, cv;
 int tmp_i;
 double tmp_d_cor, tmp_d_err, tmp_d_rej;
 FILE  *te_fp,
 *tee_fp,
```

```
*tc_fp;
FILE  *fp, *e_fp, *e4_fp,
*ee_fp, *ee4_fp,
*c_fp, *c4_fp;

if( ( argc >= 2 ) && ( *(*(argv+1)+0) == '?' ) ){
 usage();
 exit( 1 );
}

para.d_f_name = (char*)alloc_1( 20, sizeof(char) );
para.part = (char*)alloc_1( 500, sizeof(char) );
read_parafile( &para );
modify_parameter( argc, argv, &para );

srand( para.seed );

if(  !( fp = fopen( para.d_f_name, "rt" ) )  ){
 fprintf( stderr, "\nCannot Open File (%s)\n", para.d_f_name );
 exit( 1 );
}
read_data1( fp, &para );
smp = (SAMPLE*)alloc_1( para.s_num, sizeof(PARAMETER) );
for( i = 0 ; i < para.s_num ; i++ )
 (smp+i)->x = (double*)alloc_1( para.s_dim, sizeof(double) );

read_data2( fp, &para, smp );
fclose( fp );

/*===== Allocation =====*/
current = (CLASSIFIER*)alloc_1( para.N_pop, sizeof(CLASSIFIER) );
for( i = 0 ; i < para.N_pop ; i++ )
 (current+i)->condition = (CONDITION*)alloc_1( para.s_dim, sizeof(CONDITION) );

elite = (CLASSIFIER*)alloc_1( para.N_pop, sizeof(CLASSIFIER) );
for( i = 0 ; i < para.N_pop ; i++ )
 (elite+i)->condition = (CONDITION*)alloc_1( para.s_dim, sizeof(CONDITION) );

eelite = (CLASSIFIER*)alloc_1( para.N_pop, sizeof(CLASSIFIER) );
for( i = 0 ; i < para.N_pop ; i++ )
 (eelite+i)->condition = (CONDITION*)alloc_1( para.s_dim, sizeof(CONDITION) );

tee_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );
te_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );
tc_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );
e_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );
c_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );
ee_perf = (PERFORMANCE*)alloc_1( 10, sizeof(PERFORMANCE) );

for( cv = 0 ; cv < 10 ; cv++ ){
 /*===== Algorithm =====*/

 /* Initialization of Classifiers */
 for( i = 0 ; i < para.N_pop ; i++ ){
  for( j = 0 ; j < para.s_dim ; j++ ){
   tmp_c = *(para.part+j);
                tmp_i = rand() % (atoi(&tmp_c)+1);
   if( tmp_i ){
    ((current+i)->condition+j)->part = atoi(&tmp_c);
        ((current+i)->condition+j)->posi = tmp_i;
```

```
    }
    else{
      ((current+i)->condition+j)->part = 1;
      ((current+i)->condition+j)->posi = 1;
    }
  }

  /* Decision of consequent part */
  decide_consequent( current+i, smp, &para, cv );

  (current+i)->correct = 0;
  (current+i)->wrong = 0;
  (current+i)->fitness = 0.0;
 }

/* Preprocedure */
(te_perf+cv)->correct = -100;
(te_perf+cv)->error = 1000;
(te_perf+cv)->reject = 0;

(tee_perf+cv)->correct = -100;
(tee_perf+cv)->error = 1000;
(tee_perf+cv)->reject = 0;

(tc_perf+cv)->correct = -100;
(tc_perf+cv)->error = 1000;
(tc_perf+cv)->reject = 0;

(e_perf+cv)->correct = -100;
(e_perf+cv)->error = 1000;
(e_perf+cv)->reject = 0;
e_rule.total = 0;
e_rule.disjoin = 0;

(ee_perf+cv)->correct = -100;
(ee_perf+cv)->error = 1000;
(ee_perf+cv)->reject = 0;
ee_rule.total = 0;
ee_rule.disjoin = 0;

(c_perf+cv)->correct = -100;
(c_perf+cv)->error = 1000;
(c_perf+cv)->reject = 0;
c_rule.total = 0;
c_rule.disjoin = 0;


sprintf( buf, "train%d.eli", cv );
if( !( e_fp = fopen( buf, "wt" ) ) ){
  fprintf( stderr, "\nCannot Open File( %s )\n", buf );
  exit( 1 );
}
fprintf( e_fp, "generations, correct rate\n" );
if( !( e4_fp = fopen( "rule.eli", "wt" ) ) ){
  fprintf( stderr, "\nCannot Open File( r_reject.eli )\n" );
  exit( 1 );
}
fprintf( e4_fp, "generations, reject rate\n" );

sprintf( buf, "train%d.eeli", cv );
```

```
if(  !( ee_fp = fopen( buf, "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( %s )\n", buf );
 exit( 1 );
}
fprintf( ee_fp, "generations, correct rate\n" );
if(  !( ee4_fp = fopen( "rule.eeli", "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( r_reject.eeli )\n" );
 exit( 1 );
}
fprintf( ee4_fp, "generations, reject rate\n" );

sprintf( buf, "train%d.cur", cv );
if(  !( c_fp = fopen( buf, "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( %s )\n", buf );
 exit( 1 );
}
fprintf( c_fp, "generations, correct rate\n" );
if(  !( c4_fp = fopen( "rule.cur", "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( r_reject.cur )\n" );
 exit( 1 );
}
fprintf( c4_fp, "generations, reject rate\n" );


sprintf( buf, "test%d.eli", cv );
if(  !( te_fp = fopen( buf, "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( %s )\n", buf );
 exit( 1 );
}
fprintf( te_fp, "generations, correct rate\n" );

sprintf( buf, "test%d.eeli", cv );
if(  !( tee_fp = fopen( buf, "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( %s )\n", buf );
 exit( 1 );
}
fprintf( tee_fp, "generations, correct rate\n" );

sprintf( buf, "test%d.cur", cv );
if(  !( tc_fp = fopen( buf, "wt" ) )  ){
 fprintf( stderr, "\nCannot Open File( %s )\n", buf );
 exit( 1 );
}
fprintf( tc_fp, "generations, correct rate\n" );


for( n_gen = 0 ; n_gen < (para.N_gen+1) ; n_gen++ ){

 for( i = 0 ; i <para.N_pop ; i++ ){
  decide_consequent( current+i, smp, &para, cv );
  (current+i)->correct = 0;
  (current+i)->wrong = 0;
  (current+i)->fitness = 0.0;
 }

 how_many_rules( current, &para );

 /*===== Learning of Certainty Factor =====*/
 for( i = 0 ; i <  para.N_learn ; i++ )
  learn_cf( current, smp, &para, cv );
```

```
  /*===== Evaluation =====*/
  *(c_perf+cv) = evaluate( current, smp, &para, cv );
  c_rule = how_many_rules( current, &para );
  *(tc_perf+cv) = t_evaluate( current, smp, &para, cv );

  /*===== Elite Preservation =====*/
  if( (c_perf+cv)->correct > (e_perf+cv)->correct ){
    preserve_elite( current, elite, &para );
    (te_perf+cv)->correct = (tc_perf+cv)->correct;
    (te_perf+cv)->error   = (tc_perf+cv)->error;
    (te_perf+cv)->reject  = (tc_perf+cv)->reject;
    (e_perf+cv)->correct = (c_perf+cv)->correct;
    (e_perf+cv)->error   = (c_perf+cv)->error;
    (e_perf+cv)->reject  = (c_perf+cv)->reject;
    e_rule.total = c_rule.total;
    e_rule.disjoin = c_rule.disjoin;
  }


        if( ( para.W_ncp * (c_perf+cv)->correct - para.W_nmp * (c_perf+cv)->error )
                > ( para.W_ncp * (ee_perf+cv)->correct - para.W_nmp * (ee_perf+cv)->error ) ){
                preserve_elite( current, eelite, &para );
                (tee_perf+cv)->correct = (tc_perf+cv)->correct;
                (tee_perf+cv)->error   = (tc_perf+cv)->error;
                (tee_perf+cv)->reject  = (tc_perf+cv)->reject;
                (ee_perf+cv)->correct = (c_perf+cv)->correct;
                (ee_perf+cv)->error   = (c_perf+cv)->error;
                (ee_perf+cv)->reject  = (c_perf+cv)->reject;
                ee_rule.total = c_rule.total;
                ee_rule.disjoin = c_rule.disjoin;
        }


  /*===== Output Files =====*/

  fprintf( e_fp, "%d, %d, %d, %d\n", n_gen, (e_perf+cv)->correct, (e_perf+cv)->error, (e_perf+cv)-
>reject );
  fflush( e_fp );
  fprintf( e4_fp, "%d, %d\n", n_gen, e_rule.total );
  fflush( e4_fp );

  fprintf( ee_fp, "%d, %d, %d, %d\n", n_gen, (ee_perf+cv)->correct, (ee_perf+cv)->error, (ee_perf+cv)-
>reject );
  fflush( ee_fp );
  fprintf( ee4_fp, "%d, %d\n", n_gen, ee_rule.total );
  fflush( ee4_fp );

  fprintf( c_fp, "%d, %d, %d, %d\n", n_gen, (c_perf+cv)->correct, (c_perf+cv)->error, (e_perf+cv)-
>reject );
  fflush( c_fp );
  fprintf( c4_fp, "%d, %d\n", n_gen, c_rule.total );
  fflush( c4_fp );


  fprintf( te_fp, "%d, %d, %d, %d\n", n_gen, (te_perf+cv)->correct, (te_perf+cv)->error, (te_perf+cv)-
>reject );
  fflush( te_fp );
```

```
  fprintf( tee_fp, "%d, %d, %d, %d\n", n_gen, (tee_perf+cv)->correct, (tee_perf+cv)->error,
(tee_perf+cv)->reject );
  fflush( tee_fp );

  fprintf( tc_fp, "%d, %d, %d, %d\n", n_gen, (tc_perf+cv)->correct, (tc_perf+cv)->error, (te_perf+cv)-
>reject );
  fflush( tc_fp );



  if( (n_gen%100) == 0 )
    out_rule( elite, current, n_gen, &para );


  /*===== Termination Test =====*/
  if( n_gen == para.N_gen )
    break;

  /*===== Genetic Operation =====*/
  g_o( current, &para );

 }

 fclose( te_fp );
 fclose( tee_fp );
 fclose( tc_fp );

 fclose( e_fp );
 fclose( e4_fp );
 fclose( ee_fp );
 fclose( ee4_fp );
 fclose( c_fp );
 fclose( c4_fp );

 if( !( fp = fopen( "trn-eli.res", "at" ) ) ){
  fprintf( stderr, "\nCannot Open File( trn-eli.res )\n" );
  exit( 1 );
 }
 fprintf( fp, "%d, %d, %d\n", (e_perf+cv)->correct, (e_perf+cv)->error, (e_perf+cv)->reject );
 fflush( fp );
 fclose( fp );

 if( !( fp = fopen( "trn-cur.res", "at" ) ) ){
  fprintf( stderr, "\nCannot Open File( trn-cur.res )\n" );
  exit( 1 );
 }
 fprintf( fp, "%d, %d, %d\n", (c_perf+cv)->correct, (c_perf+cv)->error, (c_perf+cv)->reject );
 fflush( fp );
 fclose( fp );

 if( !( fp = fopen( "tes-eli.res", "at" ) ) ){
  fprintf( stderr, "\nCannot Open File( tes-eli.res )\n" );
  exit( 1 );
 }
 fprintf( fp, "%d, %d, %d\n", (te_perf+cv)->correct, (te_perf+cv)->error, (te_perf+cv)->reject );
 fflush( fp );
 fclose( fp );

 if( !( fp = fopen( "tes-cur.res", "at" ) ) ){
  fprintf( stderr, "\nCannot Open File( tes-cur.res )\n" );
```

```
            exit( 1 );
    }
    fprintf( fp, "%d, %d, %d\n", (tc_perf+cv)->correct, (tc_perf+cv)->error, (tc_perf+cv)->reject );
    fflush( fp );
    fclose( fp );


}
    /*===== Output Result =====*/

    tmp_d_err = 0.0;
    tmp_d_cor = 0.0;
    tmp_d_rej = 0.0;
    for( cv = 0 ; cv < 10 ; cv++ ){
            tmp_d_err += (double)(e_perf+cv)->error;
            tmp_d_cor += (double)(e_perf+cv)->correct;
            tmp_d_rej += (double)(e_perf+cv)->reject;
    }
    tmp_d_err /= 10.0;
    tmp_d_cor /= 10.0;
    tmp_d_rej /= 10.0;
    if( !( e_fp = fopen( "result.eli", "at" ) ) ){
            fprintf( stderr, "\nCannot Open File( result.eli )\n" );
            exit( 1 );
    }
    fprintf( e_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f, rule = %d\n",
(tmp_d_cor*10.0)/(double)(para.s_num*9),
            (tmp_d_err*10.0)/(double)(para.s_num*9),
            (tmp_d_rej*10.0)/(double)(para.s_num*9), e_rule.total );
    fclose( e_fp );

    tmp_d_err = 0.0;
    tmp_d_cor = 0.0;
    tmp_d_rej = 0.0;
    for( cv = 0 ; cv < 10 ; cv++ ){
            tmp_d_err += (double)(e_perf+cv)->error;
            tmp_d_cor += (double)(e_perf+cv)->correct;
            tmp_d_rej += (double)(e_perf+cv)->reject;
    }
    tmp_d_err /= 10.0;
    tmp_d_cor /= 10.0;
    tmp_d_rej /= 10.0;
    if( !( e_fp = fopen( "training.res", "at" ) ) ){
            fprintf( stderr, "\nCannot Open File( result.eli )\n" );
            exit( 1 );
    }
    fprintf( e_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f, rule = %d\n",
(tmp_d_cor*10.0)/(double)(para.s_num*9),
            (tmp_d_err*10.0)/(double)(para.s_num*9),
            (tmp_d_rej*10.0)/(double)(para.s_num*9), e_rule.total );
    fclose( e_fp );


    tmp_d_err = 0.0;
    tmp_d_cor = 0.0;
    tmp_d_rej = 0.0;
    for( cv = 0 ; cv < 10 ; cv++ ){
            tmp_d_err += (double)(ee_perf+cv)->error;
            tmp_d_cor += (double)(ee_perf+cv)->correct;
            tmp_d_rej += (double)(ee_perf+cv)->reject;
```

```
 }
 tmp_d_err /= 10.0;
 tmp_d_cor /= 10.0;
 tmp_d_rej /= 10.0;
 if( !( ee_fp = fopen( "result.eeli", "at" ) ) ){
          fprintf( stderr, "\nCannot Open File( result.eeli )\n" );
          exit( 1 );
 }
 fprintf( ee_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f, rule = %d\n",
(tmp_d_cor*10.0)/(double)(para.s_num*9),
          (tmp_d_err*10.0)/(double)(para.s_num*9),
          (tmp_d_rej*10.0)/(double)(para.s_num*9), ee_rule.total );
 fclose( ee_fp );

 tmp_d_err = 0.0;
 tmp_d_cor = 0.0;
 tmp_d_rej = 0.0;
 for( cv = 0 ; cv < 10 ; cv++ ){
          tmp_d_err += (double)(c_perf+cv)->error;
          tmp_d_cor += (double)(c_perf+cv)->correct;
          tmp_d_rej += (double)(c_perf+cv)->reject;
 }
 tmp_d_err /= 10.0;
 tmp_d_cor /= 10.0;
 tmp_d_rej /= 10.0;
 if( !( c_fp = fopen( "result.cur", "at" ) ) ){
          fprintf( stderr, "\nCannot Open File( result.cur )\n" );
          exit( 1 );
 }
 fprintf( c_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f, rule = %d\n",
(tmp_d_cor*10.0)/(double)(para.s_num*9),
          (tmp_d_err*10.0)/(double)(para.s_num*9),
          (tmp_d_rej*10.0)/(double)(para.s_num*9), c_rule.total );
 fclose( c_fp );



 tmp_d_err = 0.0;
 tmp_d_cor = 0.0;
 tmp_d_rej = 0.0;
 for( cv = 0 ; cv < 10 ; cv++ ){
          tmp_d_err += (double)(te_perf+cv)->error;
          tmp_d_cor += (double)(te_perf+cv)->correct;
          tmp_d_rej += (double)(te_perf+cv)->reject;
 }
 tmp_d_err /= 10.0;
 tmp_d_cor /= 10.0;
 tmp_d_rej /= 10.0;
 if( !( e_fp = fopen( "result.teli", "at" ) ) ){
          fprintf( stderr, "\nCannot Open File( result.teli )\n" );
          exit( 1 );
 }
 fprintf( e_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f\n", (tmp_d_cor*10.0)/(double)para.s_num,
          (tmp_d_err*10.0)/(double)para.s_num,
          (tmp_d_rej*10.0)/(double)para.s_num );
 fclose( e_fp );


 tmp_d_err = 0.0;
 tmp_d_cor = 0.0;
```

```
tmp_d_rej = 0.0;
for( cv = 0 ; cv < 10 ; cv++ ){
        tmp_d_err += (double)(te_perf+cv)->error;
        tmp_d_cor += (double)(te_perf+cv)->correct;
        tmp_d_rej += (double)(te_perf+cv)->reject;
}
tmp_d_err /= 10.0;
tmp_d_cor /= 10.0;
tmp_d_rej /= 10.0;
if(  !( e_fp = fopen( "test.res", "at" ) )  ){
        fprintf( stderr, "\nCannot Open File( result.teli )\n" );
        exit( 1 );
}
fprintf( e_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f\n", (tmp_d_cor*10.0)/(double)para.s_num,
         (tmp_d_err*10.0)/(double)para.s_num,
         (tmp_d_rej*10.0)/(double)para.s_num );
fclose( e_fp );



tmp_d_err = 0.0;
tmp_d_cor = 0.0;
tmp_d_rej = 0.0;
for( cv = 0 ; cv < 10 ; cv++ ){
        tmp_d_err += (double)(tee_perf+cv)->error;
        tmp_d_cor += (double)(tee_perf+cv)->correct;
        tmp_d_rej += (double)(tee_perf+cv)->reject;
}
tmp_d_err /= 10.0;
tmp_d_cor /= 10.0;
tmp_d_rej /= 10.0;
if(  !( ee_fp = fopen( "result.teeli", "at" ) )  ){
        fprintf( stderr, "\nCannot Open File( result.teeli )\n" );
        exit( 1 );
}
fprintf( ee_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f\n", (tmp_d_cor*10.0)/(double)para.s_num,
         (tmp_d_err*10.0)/(double)para.s_num,
         (tmp_d_rej*10.0)/(double)para.s_num );
fclose( ee_fp );

tmp_d_err = 0.0;
tmp_d_cor = 0.0;
tmp_d_rej = 0.0;
for( cv = 0 ; cv < 10 ; cv++ ){
        tmp_d_err += (double)(tc_perf+cv)->error;
        tmp_d_cor += (double)(tc_perf+cv)->correct;
        tmp_d_rej += (double)(tc_perf+cv)->reject;
}
tmp_d_err /= 10.0;
tmp_d_cor /= 10.0;
tmp_d_rej /= 10.0;
if(  !( c_fp = fopen( "result.tcur", "at" ) )  ){
        fprintf( stderr, "\nCannot Open File( result.tcur )\n" );
        exit( 1 );
}
fprintf( c_fp, "ncp = %6.4f, error = %6.4f, reject = %6.4f\n", (tmp_d_cor*10.0)/(double)para.s_num,
         (tmp_d_err*10.0)/(double)para.s_num,
         (tmp_d_rej*10.0)/(double)para.s_num );
fclose( c_fp );
```

```
 /*===== Termination Procedure =====*/
 free( e_perf );
 free( c_perf );
 free( ee_perf );
 free( te_perf );
 free( tc_perf );
 free( tee_perf );
 free( current );
 free( elite );
 free( eelite );
 for( i = 0 ; i < para.s_num ; i++ )
   free( (smp+i)->x );
 free( smp );
 free( para.d_f_name );
 free( para.part );

 return( 0 );
}



#include"cs.h"
/*#define RAND_MAX (2147483647)*/
/*-------------------------------------------------------------*/

void usage( void )
{
 fprintf( stderr, "\n_____\n" );
 fprintf( stderr, "-------------------------\n" );
 fprintf( stderr, "   Fuzzy Classifier System      \n" );
 fprintf( stderr, "                                          \n" );
 fprintf( stderr, "_____\n" );
 fprintf( stderr, "-------------------------\n" );
 fprintf( stderr, "\nOptions..\n" );
 fprintf( stderr, "\t1[xxxxx] ... Population size\n" );
 fprintf( stderr, "\t2[0.xxx] ... Replacement proportion\n" );
 fprintf( stderr, "\t3[0.xxx] ... Mutation probability\n" );
 fprintf( stderr, "\t4[xxxxx] ... Learning iteration\n" );
 fprintf( stderr, "\t5[0.xxx] ... eta_1( learning rate )\n" );
 fprintf( stderr, "\t6[0.xxx] ... eta_2( learning rate )\n" );
 fprintf( stderr, "\t7[x.xxx] ... W_ncp( weight of fitness )\n" );
 fprintf( stderr, "\t8[x.xxx] ... W_nmp( weight of fitness )\n" );
 fprintf( stderr, "\t9[xxxxx] ... the number of generations\n" );
 fprintf( stderr, "\t0[fname] ... data file name\n" );
 fprintf( stderr, "\ta[xxxxx] ... random seed\n" );
 fprintf( stderr, "\tb[t or b]... the shape of membership function\n" );
 fprintf( stderr, "\tc[xxx..] ... the number of partitions of each axis\n" );
}

/*-------------------------------------------------------------*/

void read_parafile( PARAMETER *pa )
{
 FILE *fp;
 int i;
 int n_para;    /* number of parameters */
 char **buf;
```

```
 n_para = 13;
 buf = (char**)alloc_2( n_para, 100, sizeof(char*), sizeof(char) );

 if(  !( fp = fopen( "paramtr.dat", "rt" ) )  ){
  fprintf( stderr, "\nCannot Open File( paramtr.dat )\n" );
  exit( 1 );
 }

 for( i = 0 ; i < n_para ; i++ ){
  myread( fp, *(buf+i) );
  myread( fp, *(buf+i) );
 }

 pa->N_pop = atoi( *(buf+0) );
 pa->P_rep = atof( *(buf+1) );
 pa->P_mut = atof( *(buf+2) );
 pa->N_learn = atoi( *(buf+3) );
 pa->eta_1 = atof( *(buf+4) );
 pa->eta_2 = atof( *(buf+5) );
 pa->W_ncp = atof( *(buf+6) );
 pa->W_nmp = atof( *(buf+7) );
 pa->N_gen = atoi( *(buf+8) );
 strcpy( pa->d_f_name, *(buf+9) );
 pa->seed  = atoi( *(buf+10) );
 pa->shape = *(*(buf+11)+0);
 strcpy( pa->part, *(buf+12) );

 free_2( (void**)buf, n_para );
 fclose( fp );

 return;
}

/*-------------------------------------------------------------*/

void read_data1( FILE *fp, PARAMETER *pa )
{
 int i, j;
 char *buf;

 buf = (char*)alloc_1( 100, sizeof(char) );

 myread( fp, buf );
 myread( fp, buf );
 pa->s_num = atoi( buf );
 myread( fp, buf );
 myread( fp, buf );
 pa->s_dim = atoi( buf );
 myread( fp, buf );
 myread( fp, buf );
 pa->s_cls = atoi( buf );

 free( buf );

 return;
}

/*-------------------------------------------------------------*/
```

```
void read_data2( FILE *fp, PARAMETER *pa, SAMPLE *smp )
{
 int i, j, flag, r_v, *idx;
 long int *num;
 char *buf;
 FILE *fpout;

 idx = (int*)alloc_1( 10, sizeof(int) );
 num = (long int*)alloc_1( 10, sizeof(long int) );

 for( i = 0 ; i < 10 ; i++ ){
  *(idx+i) = 0;
  *(num+i) = pa->s_num / 10;
 }

 for( i = 0 ; i < (pa->s_num%10) ; i++ )
  *(num+i) += 1;

 buf = (char*)alloc_1( 100, sizeof(char) );


 if( !( fpout = fopen( "testout", "wt" ) ) ){
        fprintf( stderr, "\ncannot open file( testout )\n" );
        exit( 1 );
 }


 for( i = 0 ; i < pa->s_num ; i++ ){

        (smp+i)->x = (double*)alloc_1( pa->s_dim, sizeof(double) );
        for( j = 0 ; j < pa->s_dim ; j++ ){
                myread( fp, buf );
                *((smp+i)->x+j) = atof( buf );
                        fprintf( fpout, "%6.4f, ", *((smp+i)->x+j) );
        }
         fprintf( fpout, "\n" );
        myread( fp, buf );
        (smp+i)->clss = atoi( buf ) - 1;
        flag = 0;
        while( flag == 0 ){
                r_v = rand() % 10;
                if( *(idx+r_v) < *(num+r_v) ){
                  (smp+i)->group = r_v;
                  *(idx+r_v) += 1;
                  flag = 1;
                 }
        }
        fprintf( fpout, "%d, %d\n", (smp+i)->clss, (smp+i)->group );
 }

        fclose( fpout );
 free( buf );
 free( idx );
 free( num );

 return;
}

/*------------------------------------------------------------*/
```

```
void modify_parameter( int ac, char **av, PARAMETER *pa )
{
 int i;

 for( i = 1 ; i < ac ; i++ ){
          if( *(*(av+i)+0) == '1' )
                    pa->N_pop = atoi( *(av+i)+1 );
          else if( *(*(av+i)+0) == '2' )
                    pa->P_rep = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '3' )
                    pa->P_mut = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '4' )
                    pa->N_learn = atoi( *(av+i)+1 );
          else if( *(*(av+i)+0) == '5' )
                    pa->eta_1 = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '6' )
                    pa->eta_2 = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '7' )
                    pa->W_ncp = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '8' )
                    pa->W_nmp = atof( *(av+i)+1 );
          else if( *(*(av+i)+0) == '9' )
                    pa->N_gen = atoi( *(av+i)+1 );
          else if( *(*(av+i)+0) == '0' )
                    strcpy( pa->d_f_name, *(av+i)+1 );
          else if( *(*(av+i)+0) == 'a' )
                    pa->seed = atoi( *(av+i)+1 );
          else if( *(*(av+i)+0) == 'b' )
                    pa->shape = *(*(av+i)+1);
          else if( *(*(av+i)+0) == 'c' )
                    strcpy( pa->part, *(av+i)+1 );
 }

 return;
}

/*-------------------------------------------------------------*/

void decide_consequent( CLASSIFIER *clfr, SAMPLE *smp, PARAMETER *pa, int cv )
{
 int i, j;
 int d_cls;          /*  decided class  */
 int flag;
 double *beta, max_b, sum_b;
 double tmp_d;

 beta = (double*)alloc_1( pa->s_cls, sizeof(double) );
 for( i = 0 ; i < pa->s_cls ; i++ )
          *(beta+i) = 0.0;

 for( i = 0 ; i < pa->s_num ; i++ ){
          if( (smp+i)->group != cv ){
          tmp_d = 1.0;
          for( j = 0 ; j < pa->s_dim ; j++ )
                    tmp_d *= membership( clfr->condition+j, *((smp+i)->x+j), pa );
          *(beta+(smp+i)->clss) += tmp_d;
          }
 }

 sum_b = 0.0;
```

```
for( i = 0 ; i < pa->s_cls ; i++ ){
        sum_b += *(beta+i);
        if( i ){
                if( max_b < *(beta+i) ){
        max_b = *(beta+i);
        d_cls = i;
        flag = 1;
                }
                else if( max_b == *(beta+i) )
        flag = 0;
        }
        else{
                max_b = *(beta+i);
                d_cls = i;
                flag = 1;
        }
}

if( flag ){
        /*  decided class  */
        clfr->clss = d_cls;
        /*  certainty factor  */
        tmp_d =  ( sum_b - *(beta+d_cls) ) / ( (double)pa->s_cls - 1.0 );
        clfr->cf = ( *(beta+d_cls) - tmp_d ) / sum_b;
}
else{
        /*  dummy class  */
        clfr->clss = -1;
        clfr->cf = 0.0;
}

free( beta );
return;
}

/*----------------------------------------------------------------*/

PERFORMANCE evaluate( CLASSIFIER *clfr, SAMPLE *smp, PARAMETER *pa, int cv )
{
 int i, j, k;
 int max_index, flag;
 double alpha, max_a;
 PERFORMANCE result;

 result.correct = 0;
 result.error   = 0;
 result.reject  = 0;

 for( i = 0 ; i < pa->s_num ; i++ ){
        if( (smp+i)->group != cv ){
        for( j = 0 ; j < pa->N_pop ; j++ ){
                alpha = (clfr+j)->cf;
                for( k = 0 ; k < pa->s_dim ; k++ )
        alpha *= membership( (clfr+j)->condition+k, *((smp+i)->x+k), pa );

                if( j ){
        if( max_a < alpha ){
          max_a = alpha;
          max_index = j;
          flag = 1;
```

```
        }
        else if( ( max_a == alpha ) && ( (clfr+max_index)->clss != (clfr+j)->clss ) )
         flag = 0;
                }
                else{
        max_a = alpha;
        max_index = j;
        flag = 1;
                }
         }

         if( flag && (max_a!=0.0) ){
                if( (clfr+max_index)->clss == (smp+i)->clss ){
        /*  Correctly Classified  */
        (clfr+max_index)->correct++;
        (clfr+max_index)->fitness += pa->W_ncp;
        result.correct++;
                }
                else{
        /*  Wrongly Classified  */
        (clfr+max_index)->wrong++;
        (clfr+max_index)->fitness -= pa->W_nmp;
        result.error++;
                }
         }
         else{
                /*  Classification rejected  */
                result.reject++;
         }
         }
 }

 return( result );
}

/*-------------------------------------------------------------*/

PERFORMANCE t_evaluate( CLASSIFIER *clfr, SAMPLE *smp, PARAMETER *pa, int cv )
{
 int i, j, k;
 int max_index, flag;
 double alpha, max_a;
 PERFORMANCE result;

 result.correct = 0;
 result.error   = 0;
 result.reject  = 0;

 for( i = 0 ; i < pa->s_num ; i++ ){
         if( (smp+i)->group == cv ){
         for( j = 0 ; j < pa->N_pop ; j++ ){
                 alpha = (clfr+j)->cf;
                 for( k = 0 ; k < pa->s_dim ; k++ )
        alpha *= membership( (clfr+j)->condition+k, *((smp+i)->x+k), pa );

                 if( j ){
        if( max_a < alpha ){
          max_a = alpha;
          max_index = j;
          flag = 1;
```

```
        }
        else if(  ( max_a == alpha ) && ( (clfr+max_index)->clss != (clfr+j)->clss ) )
          flag = 0;
                }
                else{
        max_a = alpha;
        max_index = j;
        flag = 1;
                }
         }

         if( flag && (max_a!=0.0) ){
                if( (clfr+max_index)->clss == (smp+i)->clss ){
        /*  Correctly Classified  */
        (clfr+max_index)->correct++;
        (clfr+max_index)->fitness += pa->W_ncp;
        result.correct++;
                }
                else{
        /*  Wrongly Classified  */
        (clfr+max_index)->wrong++;
        (clfr+max_index)->fitness -= pa->W_nmp;
        result.error++;
                }
         }
         else{
                /*  Classification rejected  */
                result.reject++;
         }
         }
 }

 return( result );
}

/*-------------------------------------------------------------*/

void learn_cf( CLASSIFIER *clfr, SAMPLE *smp, PARAMETER *pa, int cv )
{
 int i, j, k;
 int max_index, flag;
 double alpha, max_a;

 for( i = 0 ; i < pa->s_num ; i++ ){
         if( (smp+i)->group != cv ){
         for( j = 0 ; j < pa->N_pop ; j++ ){
                 alpha = (clfr+j)->cf;
                 for( k = 0 ; k < pa->s_dim ; k++ )
        alpha *= membership( (clfr+j)->condition+k, *((smp+i)->x+k), pa );

                if( j ){
        if( max_a < alpha ){
          max_a = alpha;
          max_index = j;
          flag = 1;
        }
        else if(  ( max_a == alpha ) && ( (clfr+max_index)->clss != (clfr+j)->clss ) )
          flag = 0;
                }
                else{
```

```
            max_a = alpha;
            max_index = j;
            flag = 1;
                    }
             }

             if( flag && (max_a!=0.0) ){
                    if( (clfr+max_index)->clss == (smp+i)->clss ){
            /*  Correctly Classified  */
            (clfr+max_index)->cf = (clfr+max_index)->cf +
             pa->eta_1 * ( 1.0 - (clfr+max_index)->cf );
                    }
                    else{
            /*  Wrongly Classified  */
            (clfr+max_index)->cf = (clfr+max_index)->cf -
             pa->eta_2 * (clfr+max_index)->cf;
                    }
             }
             }

 }

 return;

}

/*-------------------------------------------------------------*/

I_SET how_many_rules( CLASSIFIER *clfr, PARAMETER *pa)
{
 int i, j, k, flag;
 I_SET result;

 result.total = 0;
 result.disjoin = 0;
 for( i = 0 ; i < pa->N_pop ; i++ ){
          if( (clfr+i)->correct || (clfr+i)->wrong )
                    result.total++;


             for( j = (i+1) ; j < pa->N_pop ; j++ ){
                    flag = 0;
                    for( k = 0 ; k < pa->s_dim ; k++ ){
            if( (((clfr+i)->condition+k)->part == ((clfr+j)->condition+k)->part) &&
                     (((clfr+i)->condition+k)->posi == ((clfr+j)->condition+k)->posi) )
              flag++;
                    }
                    if( flag == pa->s_dim ){
            flag = 0;
            (clfr+i)->cf = 0.0;
            break;
                    }
                    else
            flag = 1;
             }
             if( flag )
                    result.disjoin++;
 }

 return( result );
```

```
}

/*-------------------------------------------------------------*/

void preserve_elite( CLASSIFIER *cur, CLASSIFIER *eli, PARAMETER *pa )
{
        int i, j;

        for( i = 0 ; i < pa->N_pop ; i++ ){
                for( j = 0 ; j < pa->s_dim ; j++ ){
                        ((eli+i)->condition+j)->part = ((cur+i)->condition+j)->part;
                        ((eli+i)->condition+j)->posi = ((cur+i)->condition+j)->posi;
                }
                (eli+i)->clss = (cur+i)->clss;
                (eli+i)->cf   = (cur+i)->cf;
                (eli+i)->fitness = (cur+i)->fitness;
                (eli+i)->correct = (cur+i)->correct;
                (eli+i)->wrong   = (cur+i)->wrong;
        }

        return;
}

/*-------------------------------------------------------------*/

void g_o( CLASSIFIER *clfr, PARAMETER *pa )
{
 int i;
 int p1, p2;
 int n_chldn;    /* number of children */
 double *fit;    /* fitness */
 double tmp_d, tmp_d2;

 CLASSIFIER *new, dummy;

 fit = (double*)alloc_1( pa->N_pop, sizeof(double) );

 /* == Normalization of Fitness == */

 tmp_d2 = 0.0;      /* sum of fitness */
 for( i = 0 ; i < pa->N_pop ; i++ ){
         tmp_d2 += (clfr+i)->fitness;
         if( i ){
                 if( tmp_d > (clfr+i)->fitness )
         tmp_d = (clfr+i)->fitness;
         }
         else
                 tmp_d = (clfr+i)->fitness;
 }
 if( (tmp_d2 == 0.0) || (tmp_d2 == pa->N_pop * tmp_d) ){
         for( i = 0 ; i < pa->N_pop ; i++ )
                 *(fit+i) = 1.0 / (double)pa->N_pop;
 }
 else{
         for( i = 0 ; i < pa->N_pop ; i++ )
                 *(fit+i) = ( (clfr+i)->fitness - tmp_d ) / ( tmp_d2 - pa->N_pop * tmp_d );
 }

 /* == Initialization of New Classifiers == */
 n_chldn = (int)( pa->N_pop * pa->P_rep );
```

```
 new = (CLASSIFIER*)alloc_1( n_chldn, sizeof(CLASSIFIER) );
 for( i = 0 ; i <  n_chldn ; i++ )
        (new+i)->condition = (CONDITION*)alloc_1( pa->s_dim, sizeof(CONDITION) );
 dummy.condition = (CONDITION*)alloc_1( pa->s_dim, sizeof(CONDITION) );

 /* == Genetic Operation == */
 i = 0;
 while( i < ( n_chldn-1 ) ){
        selection( fit, pa->N_pop, &p1, &p2 );
        crossover( clfr+p1, clfr+p2, new+i, new+i+1, pa->s_dim );
        mutation( new+i, pa );
        mutation( new+i+1, pa );
        i += 2;
 }
 if( (n_chldn%2) ){
        selection( fit, pa->N_pop, &p1, &p2 );
        crossover( clfr+p1, clfr+p2, new+i, &dummy, pa->s_dim );
        mutation( new+i, pa );
 }

 /* == Replacement == */
 replace( clfr, new, pa, n_chldn );

 free( fit );
 for( i = 0 ; i < n_chldn ; i++ )
        free( (new+i)->condition );
 free( new );
 free( dummy.condition );
 return;
}

/*-------------------------------------------------------------*/

void selection( double *fit, int pop, int *p1, int *p2 )
{
        int i;
        double rv;      /*  random value  */

        rv = (double)( rand()%32767 ) / 32767.0;
        for( i = 0 ; i < pop ; i++ ){
                rv -= *(fit+i);
                if( rv <= 0.0 ){
                        *p1 = i;
                        break;
                }
        }
        if( rv > 0.0 ){
                fprintf( stderr, "\nstrange random value( selection )\n" );
                exit( 1 );
        }
        rv = (double)( rand()%32767 ) / 32767.0;
        for( i = 0 ; i < pop ; i++ ){
                rv -= *(fit+i);
                if( rv <= 0.0 ){
                        *p2 = i;
                        break;
                }
        }
        if( rv > 0.0 ){
                fprintf( stderr, "\nstrange random value( selection )\n" );
```

```
                exit( 1 );
        }

        return;
}

/*-------------------------------------------------------------*/

void crossover( CLASSIFIER *p1, CLASSIFIER *p2, CLASSIFIER *n1,
                                                        CLASSIFIER *n2, int dim )
{
        int i;

        /*== block change( uniform crossover ) ==*/

        for( i = 0 ; i < dim ; i++ ){
                if( (rand()%2) ){
                        (n1->condition+i)->part = (p1->condition+i)->part;
                        (n1->condition+i)->posi = (p1->condition+i)->posi;
                        (n2->condition+i)->part = (p2->condition+i)->part;
                        (n2->condition+i)->posi = (p2->condition+i)->posi;
                }
                else{
                        (n1->condition+i)->part = (p2->condition+i)->part;
                        (n1->condition+i)->posi = (p2->condition+i)->posi;
                        (n2->condition+i)->part = (p1->condition+i)->part;
                        (n2->condition+i)->posi = (p1->condition+i)->posi;
                }
        }

        return;
}

/*-------------------------------------------------------------*/

void mutation( CLASSIFIER *clfr, PARAMETER *pa )
{
 int i;
 int tmp_i;
 char tmp_c;
 double rv;     /*  random value  */


 for( i = 0 ; i < pa->s_dim ; i++ ){
          rv = (double)( rand() % RAND_MAX ) / (double)RAND_MAX;
  if( rv < pa->P_mut ){
   tmp_c = *(pa->part+i);
   tmp_i = rand() % (atoi(&tmp_c)+1);
   if( tmp_i ){
        (clfr->condition+i)->part = atoi(&tmp_c);
        (clfr->condition+i)->posi = tmp_i;
    }
    else{
        (clfr->condition+i)->part = 1;
        (clfr->condition+i)->posi = 1;
    }
   }
 }

 return;
```

```
}

/*-------------------------------------------------------------*/

void replace( CLASSIFIER *clfr, CLASSIFIER *new, PARAMETER *pa, int n_chl )
{
        int i, j;
        int index;
        int *history;

        history = (int*)alloc_1( n_chl, sizeof(int) );

        for( i = 0 ; i < n_chl ; i++ ){
                index = select_rep( i, history, clfr, pa->N_pop );

                for( j = 0 ; j < pa->s_dim ; j++ ){
                        ((clfr+index)->condition+j)->part = ((new+i)->condition+j)->part;
                        ((clfr+index)->condition+j)->posi = ((new+i)->condition+j)->posi;
                }
        }

        free( history );
        return;
}

/*-------------------------------------------------------------*/

int select_rep( int num, int *hist, CLASSIFIER *clfr, int pop )
{
        int i, j;
        int flag;
        int result;         /*  index  */
        double  min_fit;    /*  minimum fitness  */

        min_fit = 100000.0;
        for( i = 0 ; i < pop ; i++ ){
                flag = 1;
                for( j = 0 ; j < num ; j++ ){
                        if( *(hist+j) == i ){
                                flag = 0;
                                break;
                        }
                }
                if( flag ){
                        if( min_fit > (clfr+i)->fitness ){
                                min_fit = (clfr+i)->fitness;
                                result = i;
                        }
                }
        }

        *(hist+num) = result;

        return( result );
}

/*-------------------------------------------------------------*/

void out_rule( CLASSIFIER *cur, CLASSIFIER *eli, int gen, PARAMETER *pa )
{
```

```
 int   i, j, tmp_i;
 FILE  *fp_c, *fp_e;
 char  *name_c, *name_e;

 name_c = (char*)alloc_1( 20, sizeof(char) );
 name_e = (char*)alloc_1( 20, sizeof(char) );

 tmp_i = gen / 10;
 sprintf( name_c, "rule%d.cur", tmp_i );
 sprintf( name_e, "rule%d.eli", tmp_i );
 if(  !( fp_c = fopen( name_c, "wt" ) )  ){
  fprintf( stderr, "\nCannot Open File( %s )\n", name_c );
  exit( 1 );
 }
 fprintf( fp_c, "antecedent part... , class, cf, learned cf, correct, error, reject\n" );
 if(  !( fp_e = fopen( name_e, "wt" ) )  ){
  fprintf( stderr, "\nCannot Open File( %s )\n", name_e );
  exit( 1 );
 }
 fprintf( fp_e, "antecedent part... , class, cf, correct, error, fitness\n" );

 for( i = 0 ; i < pa->N_pop ; i++ ){
  for( j = 0 ; j < pa->s_dim ; j++ ){
   fprintf( fp_c, "%d, %d\n", ((cur+i)->condition+j)->part, ((cur+i)->condition+j)->posi );
   fprintf( fp_e, "%d, %d\n", ((eli+i)->condition+j)->part, ((eli+i)->condition+j)->posi );
  }
  fprintf( fp_c, "%d, %6.4f\n", (cur+i)->clss+1, (cur+i)->cf );
  fprintf( fp_e, "%d, %6.4f\n", (eli+i)->clss+1, (eli+i)->cf );
  fprintf( fp_c, "cor = %d, err = %d, fit = %6.1f\n", (cur+i)->correct, (cur+i)->wrong, (cur+i)->fitness );
  fprintf( fp_e, "cor = %d, err = %d, fit = %6.1f\n", (eli+i)->correct, (eli+i)->wrong, (eli+i)->fitness );
 }

 fclose( fp_c );
 fclose( fp_e );
 free( name_c );
 free( name_e );

 return;
}

/*------------------------------------------------------------*/




#include"cs.h"

/*------------------------------------------------------------*/

void **alloc_2( int n1, int n2, int s1, int s2 )
{
 void **result;
 int i;

 if(  !( result = (void**)calloc( n1, s1 ) )  ){
  fprintf( stderr, "\nMemory Allocation Error\n" );
  exit( 1 );
 }
```

```
 for( i = 0 ; i < n1 ; i++ ){
  if(  !( *(result+i) = calloc( n2, s2 ) )  ){
   fprintf( stderr, "\nMemory Allocation Error_2\n" );
   exit( 1 );
  }
 }

 return( result );
}

/*-------------------------------------------------------------*/

void myread( FILE *fp, char *buf )
{
 int i, c;

 for( i = 0 ; i < 100 ; i++ ){
  c = getc( fp );

  if( ( c == '\n' ) || ( c == ',' ) )
   break;
  *(buf+i) = (char)c;
 }

 if( i == 100 ){
  fprintf( stderr, "\nInvalid contents in file\n" );
  exit( 1 );
 }
 *(buf+i) = '\0';

 return;
}

/*-------------------------------------------------------------*/

void free_2( void** obj, int n1 )
{
 int i;

 for( i = 0 ; i < n1 ; i++ )
  free( *(obj+i) );

 free( obj );

 return;
}

/*-------------------------------------------------------------*/

void *alloc_1( int n, int s )
{
 void *result;

 if(  !( result = calloc( n, s ) )  ){
  fprintf( stderr, "\nMemory Allocation Error\n" );
  exit( 1 );
 }

 return( result );
}
```

```
/*-------------------------------------------------------------*/

double membership( CONDITION *cdn, double x, PARAMETER *pa )
{
 double a, dum_b;
 double tmp_d, result;

 if( cdn->part == 1 )
  return( 1.0 );

 a = ( (double)cdn->posi - 1.0 ) / ( (double)cdn->part - 1.0 );

 if( pa->shape == 't' ){

  /*  trianglar shape  */
  dum_b = (double)cdn->part - 1.0;
  tmp_d = ( x - a );
  if( tmp_d >= 0.0 )
   result = 1.0 - tmp_d * dum_b;
  else
   result = 1.0 + tmp_d * dum_b;

  if( result <= 0.0 )
   result = 0.0;
 }
 else if( pa->shape == 'b' ){

  /*  bell shape  */
  dum_b = 4.0 * log( 2.0 ) * ( (double)cdn->part - 1.0 ) * ( (double)cdn->part - 1.0 );
  tmp_d = ( x - a ) * ( x - a ) * dum_b;
  result = exp( -tmp_d );
 }

 return( result );
}

/*-------------------------------------------------------------*/
```

-- Population size (num. of rules) --
10
-- Replacement rate --
0.2
-- Probability of mutation --
0.1
-- Num. of learnings of certainty factor --
0
-- Learning rate (eta_1) --
0.001
-- Learning rate (eta_2) --
0.1
-- Weight (in the fitness function) w_ncp --
1.0
-- Weight (in the fitness function) w_nmp --
5.0
-- Num. of generations --
100
-- Data file name --
CAD.dat
-- random seed --
10
-- shape of fuzzy sets (triangle or bell) --
triangle
-- num. of partition of each axis --
55555555555555555555

Jain, R. and Mazumdar, J. (1998) Neural network approach for the classification of heart disease data.
Australian Journal of Intelligent Information Processing Systems, v. 5 (2), pp. 86-93, Winter 1998

Jain, R., Mazumdar, J. and Moran, W. (1998) Application of fuzzy-classifier system to coronary artery disease and breast cancer.
Australasian Physical & Engineering Sciences in Medicine, v. 21 (3), pp. 141-147, September 1998

Jain, R., Mazumdar, J. and Moran, W. (1998) Application of fuzzy-classifier system to coronary artery disease and breast cancer.
Biomedical Fuzzy and Human Sciences, v. 2 (1), pp. 45-54, August 1996

Bibliography

[1] Ackley, D., Hinton, G. and Sejnowski, T., "A Learning Algorithm for Boltzmann Machines." *Cognitive Science*, 9, pp. 147-169, (1985).

[2] Adam, J.A., Medical Electronics, *IEEE Spectrum*, vol. 33, No. 1, pp. 92-95, (1996).

[3] Adlassing, K.P., "A Survey on Medical Diagnosis and Fuzzy subsets." *Approximate reasoning in Decision Analysis*, pp. 203-217, (1982).

[4] Adlassing, K.P., "Fuzzy set theory in Medical Diagnosis." *IEEE, Trans. Systems, Man and Cybernatics*, vol. 16, no. 2, pp. 260-265, (1986).

[5] Akay, M., "Non-invasive diagnosis of Coronary artery disease a neural network algorithm." *Bio. Cybernetics*, 67, pp. 361-367, (1992).

[6] Akay, M., et.al., "Neural Networks for the Diagnosis of Coronary Artery Disease." *Proceedings of first International Joint Conference on Neural Networks*, pp. 419-424, (1992).

[7] Almeida, L., "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment." *Proceedings of the IEEE first International Conference on Neural Networks*: vol 11, pp. 609-618, (1987).

[8] Alpaslan, F.N. and Jain, R. K., "Research in Biomedical sciences." *Proceedings Electronic Technology Directions to the year 2000,* pp. 612-619, (1995).

[9] Alvey, P., and Greaves, M., "A High performance system for leukaemia Diagnosis." unpublished work.

[10] Appolloni, B., et.al., "Diagnosis of Epilepsy via Backpropagation." *Applications Track of the Proceedings of the Joint Conference on Neural Networks*, vol. 2, pp. 571-574, (1990).

[11] Arita, S., et.al., "Multistage Process of Fuzzy Inference and its Application to the Medical Diagnosis." *Proceedings of the Third International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 323-324, (1994).

[12] Arita, S., et.al., "Supporting System for Medical Diagnosis Using Natural Language by a Fuzzy Inference." *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks*, pp. 1017-1019, (1992).

[13] Arnold, W.R. and Bowie, J.S., *Artificial Intelligence*. Prentice-Hall, U.S.A., (1986).

[14] Babri, H., Chen, L., Saratchandran, P., Mital D.P., Jain R. K., Johnson, R. P., "Neural Networks paradigms." *Soft computing techniques in Knowledge-based intelligent engineering systems*, Springer Verlag, Germany, Chapter 2, pp. 15-43, (1997).

[15] Bala, J., et.al. "Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification." *Proceedings of the Second International Joint Conference on Artificial Intelligence,* pp. 719-724, (1995)

[16] Baldi, P., "Gradient Descent Learning Algorithm Overview: A General Dynamical Systems Perspective.", *IEEE Trans. Neural Networks*, vol. 6, pp. 182-195, (1995).

[17] Baum, E.B. and Haussler, D., "What Size Net Gives Valid Generalisation?", *Neural Computation*, vol. 1, pp. 151-160, (1989).

[18] Baxt, W.G., "Analysis of the Clinical Variables Driving Decision in an Artificial Neural Network Trained to Identify the Presence of Myocardial Infarction." *Annals of Emergency Medicine*, 21(12), pp. 35-40, (1992).

[19] Baxt, W.G., "Use of an Artificial Neural Network for Data Analysis of Clinical Decision-Making: The Diagnosis of Acute Coronary Occlusion." *Neural Computation*, 2, pp. 480-489, (1990).

[20] Bernold, T., (Editor), *Expert Systems and Knowledge Engineering*. North-Holland, (1986).

[21] Bielawski, L. and Lewand, R., *Expert Systems Development*. QED Information Sciences Inc., Wellesley, Massachusetts, (1988).

[22] Blumenfeld, B., "A Connectionist Approach to the Processing of Time Dependent Medical Parameters." *Applications Track of the Proceedings of the International Joint Conference on Neural Networks*, vol. II, pp. 575-578, (1990).

[23] Boone, J.M., et.al., "Computer-Aided Radiologic Diagnosis Using Neural Networks." *Applications Track of the Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 98-101, (1990).

[24] Boron, G., "A flexible rule-based system for the interpretation of thyroid function tests", unpublished work.

[25] Bounds, D.G., et.al., "A Comparison of Neural Network and other Pattern Recognition Approaches to the Diagnosis of Low Back Disorders." *Neural Networks*, 3, pp. 583-591, (1990).

[26] Broman, H., "Knowledge-Based Signal Processing in the Decomposition of Myoelectric Signals." *IEEE Engineering in Medicine and Biology Magazine*, pp. 24-28, (1993).

[27] Broomhead, S. and Lowe, D., "Multivariable Functional Interpolation and Adaptive Networks." *Complex Systems*, 2, pp. 321-355, (1988).

[28] Brown, D.G., "Neural Networks for Second-Order Medical Tasks." *Proceedings of the International Joint Conference on Neural Networks*, vol. I, pp. 195-198, (1991).

[29] Brown, T.H., Zador, A. M., Mainen, Z. F. and Claiborne, B. J., "Hebbian Computations in Hippocampal Dendrites and Spines." in *Single Neuron Computation* (Editors) T. McKenna et. al., Academic Press, Inc. New York., pp. 81-116, (1988).

[30] Carolla, A., et.al., "Relief of Postoperative Pain." *Proceedings of the 1993 IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 793-796, (1993).

[31] Carollo, A., et.al., "Artificial Neural Nets in Computer-Aided Macro Motor Unit Potential Classification." *IEEE Engineering in Medicine and Biology*, pp. 31-38, (1993).

[32] Carpenter, G. A. and Grossberg, S. and Rosen D.B., "Fuzzy ART: An adaptive Resonance Algorithm for Rapid, Stable Classification of Analog Patterns." *IEEE, International Joint Conference on Neural Networks*, vol. 2, pp. 411-416, (1991b).

[33] Carpenter, G. A. and Grossberg, S., "ART2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns." *Applied Optics*, 26(23): pp. 4919-4930, (1987).

[34] Carpenter, G. A. and Grossberg, S., "ART3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures." *Neural Networks*, 3(4), pp. 129-152, (1990).

[35] Carpenter, G. A. and Grossberg, S., "The ART of Adaptive Pattern Recognition by Self-organizing Neural Network." *Computer*, 21(3), pp. 77-88, (1988).

[36] Carpenter, G. A. and Grossberg, S., and Reynolds, J., "ARTMAP: A self-organizing NN Architecture for Fast Supervised Learning and Pattern Recognition." *Proceedings of International Joint Conference on Neural Networks*, pp. 863-868, (1991a).

[37] Carpenter, G. A. and Ross, W. D., "ART-EMAP: A Neural Network Architecture for Object Recognition by Evidence Accumulation." *IEEE Trans. Neural Networks*, vol. 6 No. 4, pp. 805-818, (1995).

[38] Carpenter, G.A. Grossberg, S. Markuzon, N., Reynolds, J.H. and Rosen, D.B., "Fuzzy ARTMAP: a Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps." *IEEE Trans. Neural Networks*, vol. 3, No. 5. (1992).

[39] Cheng, K., et.al., "A Subband Coding Scheme and the Bayesian Neural Network for EMG Function Analysis." *Proceedings of the International Joint Conference Neural Networks*, vol. 2, pp. 931-934, (1992).

[40] Choi, J.J., et.al., "Fuzzy parameter adaptation in neural systems." *Proceedings of the International Joint Conference Neural Networks*, vol.1, pp. 232-238, (1992).

[41] Cios, K.J., et.al., "Use of Neural Networks in Detecting Cardiac Diseases from Echocardiographic Images." *IEEE Engineering in Medicine and Biology*, pp. 58-60, (1990).

[42] Cios, K.J., et.al., "Using Fuzzy Sets to Diagnose Coronary Artery Stenois." *IEEE Computer*, pp. 57-63, (1991).

[43] De Roach, J.N., "Neural Networks - An Artificial Intelligence Approach to the Analysis of Clinical Data." *Australasian Physical & Engineering Sciences in Medicine*, 12(2), pp. 100-106, (1989).

[44] Derthick, M. and Tebilskis, J., "Ensemble Boltzmann Units have Collective Computational properties like those of Hopfield and Tank Neurons." *Proceedings 1987 IEEE Conference on Neural Information Processing Systems - Natural and Synthetic*, pp. 223-232, (1988).

[45] DeSilva, C.J.S. et al., "Artificial Neural networks and Breast Cancer Prognosis." *The Australian Computer Journal* , vol. 26, pp. 78-81, (1994).

[46] Dhavan, A.P. and Arata, L., "Segmentation of Medical Images Through Competitive Learning."*Proceedings of the IEEE International Conference on Neural Networks*, pp. 1277-1282, (1993).

[47] Dietterich, T., "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Neural Computation*, 10, pp. 1895-1923, (1998).

[48] Drago, G. P. and Ridella, S., "Statistically Controlled Activation Weight Initialization." *IEEE Trans. Neural Networks*, vol. 3, pp. 627-631, (1992).

[49] Drucker, H. and Le Cun, Y., "Improving Generalization Performance Using Double Backpropagation." IEEE Trans. Neural Networks, vol. 3, pp. 991-997, (1992).

[50] Duda, R.O. and Hart, P.E., *Pattern Classification and Scene Analysis.* Wiley-International (1973).

[51] Ebell, M.H., "Artificial Neural Networks for Predicting Failure to Survive Following In-Hospital Cardiopulmonary Resuscitation." *The Journal of Family Practice*, 36(3), pp.297-303, (1993).

[52] Edenbrandt, L., et.al., "Classification of Electrocardiographic ST-T Segments: Human Expert vs Artificial Intelligence." *European Heart Journal*, 14, pp. 464-468, (1993).

[53] Edenbrant, L., et.al., "Neural Networks for Classification of ECG ST-T Segments." *Journal of Electrocardiology*, 25(3), 167-173, (1992).

[54] Elman, J. L., "Finding Structures in Time", *Technical Report CRL-8801*, Centre for Research in Language, University of California at San Diego (1988).

[55] Errington, P.A. and Graham, J., "Classification of Chromosomes Using a Combination of Neural Networks." *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1236-1241, (1993).

[56] Esogbue, A.O,and Elder, R.C., "Fuzzy sets and the modelling of Physician Decision Processes - Part I." *Fuzzy Sets and Systems* 2, pp. 279-291, (1979).

[57] Esogbue, A.O., "Measurements and Valuation of a Fuzzy Mathmatical model for Medical diagnosis." *Fuzzy Sets and Systems*, pp. 223-242, (1983).

[58] Ezquerra, N. and Mullick, R., "PERFEX: An Expert System for Interpreting 3D Myocardial Perfusion." *Expert Systems with Applications*, 6(4), pp. 459-468, (1993).

[59] Farrugia, S., et.al., "Neural Network Classification of Intracardiac ECGs." *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, 1278-1283, (1991).

[60] Fieschi, M., "The Sphinx project, Faculte de Medecine de Marseille France." unpublished work.

[61] Firebaugh, M.W., *Artificial intelligence.* PWS-Kent Publishing Company, Boston, USA, (1988).

[62] Forsyth, R., (Editor), *Expert Systems*. Chapman & Hall, U.S.A., (1984).

[63] Freeman, J.A. and Skapura, *D.M., Neural Networks*. Addison-Wesly publishing company, USA, (1992).

[64] Fujita, H., et.al., "Neural Network Approach for Computer-Aided Diagnosis of Coronary Artery Diseases in Nuclear Medicine." *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 215-220, (1992).

[65] Fukunaga, K, *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edn., 1990.

[66] Fukushima, K., "Neocognitron: a Heirarchical Neural Network Capable of Visual Pattern Recognition." *Neural Networks*, vol. 1, pp. 119-130, (1988).

[67] Fukushima, K., Miyake, S. and Ito, T., "Neocognitron: a Neural Network Model for a Mechanism of Visual Pattern Recognition." *IEEE Trans. Systems, Man and Cybernetics*, 13(5), pp. 826-834, (1983).

[68] *Fuzzy inference development environment.* Aptronix, (1992).

[69] Garcia, O.N. and Chien, Y., *Knowledge-based Systems: Fundamentals and Tools*. IEEE Computer Society Press, Los Alamitos, (1991).

[70] Gero, G.S., *Artificial Intelligence in Engineering: Design, Computational Mechanics.* Publications, England, (1988).

[71] Giove, S., et.al., "An Adaptive Fuzzy Control Module for Automatic Dialysis." *Proceedings of the 8th Austrian AI Conference,* (1993).

[72] Goldberg, D.E., "Genetic and evolutionary algorithms come of age." *Communications of the ACM*, vol. 37, No. 3, pp. 113-119, (1994).
[73] Goldberg, D.E., *Genetic algorithms*. Addison-Wesley Publishing Company, (1989).
[74] Grossberg, S., (Editor), *Neural Networks and Natural Intelligence*. MIT Press, Cambridge, MA (1988).
[75] Grossberg, S., "Studies of Mind and Brain." vol. 70, *Boston Studies in the Philosophy of Science*. D. Reidel Publishing Company, Boston (1992).
[76] Guida, G. and Tasso, C., *Design and Development of Knowledge-based Systems*. John Wiley and Sons, Chichester, (1994).
[77] Guida, G. and Tasso, C., *Design and Development of Knowledge-Based System*. John Wiley and Sons, (1994).
[78] Hall, L.O., et.al., "A Comparison of Neural Network and Fuzzy Clustering Techniques in Segmenting Magnetic Resonance Images of the Brain." *IEEE Trans. Neural Networks*, 3(5), pp. 672-682, (1992).
[79] Harmon, P. and King, D., *Expert Systems*. John Wiley and Sons, (1985).
[80] Harrison, R.F., et.al., "The Early Diagnosis of Heart Attacks: A Neurocomputational Approach." *Proceedings of the International Joint Conference on Neural Networks*, vol. I, pp. 1-5, (1991).
[81] Hart, A., *Knowledge Acquisition for Expert Systems*. McGraw-Hill, (1986).
[82] Hassoun, M.H., "Electromyogram Decomposition via Unsupervised Dynamic Multi-Layer Neural Network", *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 405-412, (1992).
[83] Hayashi, Y., "Neural Expert System Using Fuzzy Teaching Input and Its Application to Medical Diagnosis." *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks*, pp. 989-993, (1992).
[84] Hayes-Roth, B., et.al., "Guardian: A Prototype Intelligent Agent for Intensive-Care Monitoring." *Artificial Intelligence in Medicine*, 4(2), pp. 165-185, (1992).
[85] Haykin, S., *Neural networks*. Macmillan College Publishing Company, (1994).
[86] Hebb, D., *Organisation of Behaviour*. New York: John Wiley & Sons, (1949).
[87] Hilera, J.R., et.al., "Neural Networks for ECG Compression and Classification." *Proceedings of the Third International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 121-124, (1994).
[88] Hinton, G., Ackley, D. and Sejnowski, T., "Boltzmann Machines: Constraint Satisfaction Networks that Learn." *Technical Report CMU-CS-84-119*, Carnegie Mellon University Dept. Computer Science (1984).
[89] Hinton, G.E., et.al., "Simulating Brain Damage." *Scientific American*, pp. 58-65, (1993).
[90] Hiraiwa, A., et.al., "EEG Topography Recognition by Neural Networks." *Engineering in Medicine and Biology*, pp. 39-42, (1990).
[91] Holdaway, R.M., "Classification of Somatosensory-Evoked Potentials Recorded from Patients with Severe Head Injuries." *IEEE Engineering in Medicine and Biology*, pp. 43-49, (1990).
[92] Holland, J.H., "Genetic algorithms." *Scientific American*, pp. 44-50, (1992).
[93] Hopfield, J. J. and Tank, T. W., "Computing with Neural Circuits: A Model." *Science* 233, pp. 625-633, (1986).
[94] Hopfield, J. J., "Neural Networks as Physical Systems with Emergent Collective Computational Capabilities." *Proceedings of National Academy of Science*, 79, pp. 2554-2558, (1982).

[95] Hu, Y.H., et.al., "Artificial Neural Network for ECG Arrhythmia Monitoring." *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 987-991, (1992).

[96] Hush, D. R., and Horne, B. G., " Progress in Supervised Neural Networks." IEEE Signal Processing Magazine, pp. 8-39, (1993).

[97] Ichimura, T., et.al., "Medical Diagnosis Using a Parallel Fuzzy Inference Performed by Neural Networks." *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks*, pp. 1003-1006, (1992).

[98] Ishibuchi, H. et.al, "A fuzzy classifier system that generates fuzzy if-then rules for pattern classification problems." *Proceedings Int. Conference Evolutionary Computat.*, vol. 2, pp. 759-764, (1995).

[99] Ishibuchi, H. et.al, "Distributed representation of fuzzy rules and its application to pattern classification." *Fuzzy Sets and System*, vol. 52, pp. 21-32, (1992).

[100] Ito, K., et.al., "Limb-Function Discrimination Using EMG Signals by Neural Network and Application to Prosthetic Forearm Control." *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 1214-1219, (1991).

[101] Iwata, A., et.al., "Real Time ECG Data Compression Using Dual Three Layered Neural Networks for Digital Holter Monitor." in *Artificial Neural Networks*, Edited by T. Kohonen, et.al., North Holland, pp. 1673-1676, (1991).

[102] Jacobs, R. A., "Increased Rates of Convergence Through Learning Rate Adaptation." *Neural Networks* 1, pp. 295-307, (1988).

[103] Jain, L. C. and Allen, G. N., "Introduction to Artificial Neural Networks." *Proceedings Electronic Technology Directions to the year 2000*, pp. 35-62, (1995).

[104] Jain, L.C. and Jain, R.K. (Editors), *Hybrid Intelligent Engineering Systems.* World Scientific Publishing Company, Singapore, March (1997).

[105] Jain, R., et.al., "An Intelligent Detection of Coronary Artery Disease: A Comparative Study." *Proceedings Electronic Technology Directions to the year 2000*, pp. 113-120, (1990).

[106] Jain, R.K., "Neural Networks in Medicine." *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pp. 369-372, (1993).

[107] Jain, R.K., Mazumdar J., "Neural Network Approach for the Classification of Heart Disease Data." *Australian Journal of Intelligent Information Processing Systems* (to appear).

[108] Jain, R.K., Mazumdar J., Moran W. "Application of Fuzzy Classifier System to Pattern Classification Problems." *Australasian Physical & Engineering Sciences in Medicine*, vol. 21, Number 3, pp. 242-247, (1998).

[109] Jain, R.K., Mazumdar, J., and Moran, W., "A Comparative Study of Artificial Intelligent Techniques in the Detection of Coronary Artery Disease." *Proceedings Electronic Technology Directions to the year 2000*, pp. 113-120, (1995).

[110] Jain, R.K., Mazumdar, J., and Moran, W., "Use of Multilayer perceptron in the classification of Coronary Artery Disease Data." *Fourth International conference on ICARCV'96*, Westin Stamford Singapore, vol 3, pp. 2247-2251, (1996).

[111] Jain, R.K., Mazumdar, J., and Moran, W.,"Expert System, Fuzzy Logic and Neural Networks in the Detection of Coronary Artery Disease." *Journal of Biomedical Fuzzy and Human Sciences*,vol.2, pp. 45-54, (1996).

[112] Jansen, B.H., "Artificial Neural Nets for K-Complex Detection." *IEEE Engineering in Medicine and Biology*, pp. 50-57, (1990).

[113] Janson, D.J. and Frenzel, J.F., "Training product unit neural networks with genetic algorithms." *IEEE Expert*, pp. 26-33, (1993).

[114] Jones, B.F., "Measuring People.", *Engineering Science and Education Journal*, pp. 283-287, (1992).

[115] Jordan, M. I., "Attractor Dynamics and Parallelism in Connectionist Sequential Machine." *Proceedings of the 1986 Cognitive Science Conference*, pp. 531-546, (1986).

[116] Kadirkamanathan, V. and Niranjan, M., "A Function Estimation Approach to Sequential Learning with Neural Networks." *Neural Computations*, 5, pp. 954-975, (1993).

[117] Kallio, K., et.al., "Classification of Lung Sounds by Using Self-Organizing Feature Maps." in *Artificial Neural Networks*, ed. by Kohonen, et.al., North-Holland, pp. 803-808, (1991).

[118] Karr, C.L., "Fuzzy control of an exothermic reaction using genetic algorithms." *Engineering applications of artificial intelligence*, vol. 6, No. 6, pp. 575-582, (1993).

[119] Kaufman, J.J., et.al., "A Neural Network Approach for Bone Fracture Healing Assessment." *IEEE Engineering in Medicine and Biology*, pp. 23-29, (1990).

[120] Keller, J.M., and Tahini, H., "Implementation of Conjunctive and Disjunctive Fuzzy Logic Rules with Neural Networks." *International JOURNAL Approximate Reasoning*, vol. 6, pp. 221-240, (1996).

[121] Kohonen, T. "Self-Organization and Associative Memory." vol. 8 of Springer Series in *Information Sciences*. Springer-Verlag, New York, (1984).

[122] Kohonen, T. *Associative Memory: A System-Theoretical Approach.* Springer-Verlag, New York, (1977).

[123] Kohonen, T., *Self-Organisation and Associative Memory.* Springer-Verlag, Berlin (1988).

[124] Kohonen, T., Barnes, G. and Chrisley, R., "Statistical Pattern recognition with Neural Networks: Benchmarking." *Proceedings IEE Conference on Neural Networks*: vol. 1 pp 61-68, (1988).

[125] Kruschke, J. K. and Movellan, J. R., "Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-Propagation Networks." *IEEE Trans. Systems, Man and Cybernetics*, vol. 21, pp. 273-280, (1991).

[126] Kulikowski, C.A., "Artificial Intelligence in Medical Consultation System." *IEEE Engineering in Medicine and Biology Magazine*, pp. 34-39, (1993).

[127] Kuncheva, L.I., et.al., "A Combination Scheme of Artificial Intelligence and Fuzzy Pattern Recognition in Medical Diagnosis." *Proceedings of the 8th Austrian AI Conference,* (1993).

[128] Lapuerta, P. et.al., "Use of Neural Networks on Predicting the Risk of Coronary Artery Disease." *Computers and Biomedical Research*, 28, 38-52, (1995).

[129] Lavrac, N., et.al., "KARDIO-E An Expert System for Electrocardiographic Diagnosis of Cardiac Arrhythmias." *Expert Systems*, 2(1), pp. 46-49, (1985).

[130] Le Cun, Y., "Generalization and Network Design Strategies." In *Connectionism in Perspective*, pp. 143- 55, Amsterdam: North-Holland (1989).

[131] Lee, C.C., "Fuzzy logic in control systems: fuzzy logic controller -Part I." *IEEE Trans. Systems, Man, and Cybernetics*, vol. 20, No. 2, pp. 404-418, (1990).

[132] Lee, C.C., "Fuzzy logic in control systems: fuzzy logic controller -Part II." *IEEE Trans. Systems, Man, and Cybernetics*, vol. 20, No. 2, pp. 419-435, (1990).

[133] Lee, Y., Oh, S. and Kim, M., "The Effect of Initial Weights on Premature Saturation in Back-propagation Learning." *International Joint Conference on Neural Networks*, vol. 1, pp. 765-770, (1991).

[134] Lippmann, R.P., "Pattern Classification Using neural Networks." *IEEE Communications Magazine* 27, pp. 47-64, (1989).

[135] Lowe, D., "Adaptive radial Basis Function Non-linearities and the Problem of Generalisation." *1st IEE International Conference on Artificial Neural Networks*, pp. 171-175, (1989).

[136] Maclin, P.S. and Dempsey, "A Neural Network to Diagnose Liver Cancer." *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1492-1497, (1993).

[137] Makino, K., et.al., "An Evaluation of hoarseness by the Fuzzy Logic." *Proceedings of the Third International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 317-318, (1994).

[138] Makino, M., et.al., "Development of Fuzzy Logic Program ProFINa and Application to Adverse Drug Effect Monitoring." *Proceedings of the Third International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 311-312, (1994).

[139] Marshall, S.,J., and Harrison, R.F., "Optimization and Training of Feedforward Neural Networks by Genetic Algorithms." *Proceedings of the Second International Conference on Neural Networks*, pp. 39-43, (1991).

[140] McCulloch, W. and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics*, 7, pp. 115-133, (1943).

[141] McSherry, D. and Fullerton, K., "Preceptor: A shell for Medical Expert Systems and its Applicatons in a study of Prognostic Indices in Stroke." *Expert Systems*, 2(3), pp. 140-146, (1985).

[142] Medsker, L.R., *Hybrid Neural Network and Expert Systems.* Kluwer Academic Publishers, (1994).

[143] Mehra, P. and Wah, B.W. (Editors), *Artificial Neural Networks : Concepts and Theory.* IEEE Computer Society Press (1992).

[144] Meier, R., et.al., "Fuzzy Logic Control of Blood Pressure During Anesthesia." *IEEE Control Systems Magazine*, pp. 12-17, (1992).

[145] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs.* 2nd extended edition, Springer-Verlag, (1994).

[146] Miller, A.S., et.al., "Review of Neural Network Applications in Medical Imaging and Signal Processing." *Medical and Biology Engineering and Computing*, pp. 450-463, (1992).

[147] Miller, G.,F., et.al., "Designing Neural Networks using Genetic Algorithms." *Proceedings of the third International Conference on Genetic Algorithms*, pp. 379-384, (1989).

[148] Minsky, M. and Papert, S., *Perceptrons* Cambridge: MIT Press (1969).

[149] Montana, D.J. and Davis, L., "Training Feedforward Neural Networks Using Genetic Algorithms." *Proceedings of the Internatinal Conference on Artificial Intelligence*, pp. 762-767, (1989).

[150] Moody, J.E. and Darken, C.J., "Fast Learning in Networks of Locally-tuned Processing Units." *Neural Computation*, 1, 281-294, (1989).

[151] Morrison, M. and Attikiouzel, Y., "An Introduction to the Segmentation of Magnetic Resonance Medical Images." *The Australian Computer Journal*, 26(3), pp. 90-98, (1994).

[152] Motoki, N. and Suzuki, Y., "Recognition of P-wave, QRS-wave, a and T-wave in Electrocardiogram with Neural Networks." *Proceedings of the Second International Conference on Fuzzy Logic & Neural Networks*, pp. 1065-1068, (1992).

[153] Nagi, G., "Neural Networks - Then and Now." *IEEE Trans. Neural Networks*, vol. 2, pp. 316-318, (1991).

[154] Narendra, K. S. and Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks." *IEEE Trans. Neural Networks*, vol. 1, No. 1, pp. 4-27, IEEE Press, (1990).

[155] Neural Network Classifier for Blood Testing, *Science Applications* International. Corporation (SAIC) ANS Application Note.

[156] Norris, D., et.al., "Medical Diagnosis from patient Records - A method using fuzzy Discrination and Connectivity Analysis." *Fuzzy Sets and Systems*, pp. 73-87, (1987).

[157] Ohhashi, A., et.al., "Application of a Neural Network to Automatic Gray-level Adjustment of Medical Images." *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 974-980, (1991).

[158] Papadourakis, G.M., et.al., "Use of Artificial Neural Networks for Clinical Decision - Making (Maldescensus Testis)." *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 159-164, (1992).

[159] Parker, D. "Learning logic." *TR-87*, Cambridge, MA: Center for Computational Research in Economics and Management Science, MIT, (1985).

[160] Patel, S. and Denham, M.J., "A Hybrid Neural Net/Knowledge Based Approach to EEG Analysis." *Artificial Neural Networks*, Edited by T. Kohonen, et.al., Elsevier Science Publishing Company, pp. 1665-1668, (1991).

[161] Pham, P.H., *Expert Systems in Engineering*. IFS Publications Limited, England, (1988).

[162] Pigford, D.V. and Baur, G., *Expert systems for business*. Boyd and Fraser Publishing Company, USA, (1990).

[163] Pineda, F. J., "Generalization of Back-Propagation to Recurrent Neural Networks." *Physical Review Letters*, vol. 59, No. 19, pp. 2229-2232, (1987).

[164] Platt, John, "A Resource Allocating Network for Function Interpolation." *Neural Computation*, 3, pp. 213-225, (1991).

[165] Poli, R., et.al., "A Neural Network Expert System for Diagnosing and Treating Hypertension." *IEEE Computer*, pp. 64-71, (1991).

[166] Powell, J.D., "Radial basis function for Multivariable Interpolation: A Review." In J. C. Mason and M. G. Cox (Editors) *Algorithms for Approximation of functions and data*, Oxford Uni. Press, pp. 143-167, (1987).

[167] Raeth, P.G. (Editor), *Expert Systems : A Software Methodology for Modern Applications*. IEEE Computer Society Press, (1990).

[168] Raeth, P.G. (Editor), *Expert Systems*. IEEE Computer Society Press, Los Alamitos, (1990).

[169] Rayburn, D.B., et.al., "Identification of Mediator Specific Cardiovascular Waveforms Using a Back Propagation Neural Network." *Neural Networks*, 4(4), pp. 525-530, (1991).

[170] Reggia, J.A., "Neural Computation in Medicine." *Artificial Intelligence in Medicine*, 5, pp. 143-157, (1993).

[171] Reyna, E., Specht, D.F. and Lee, A., "Small, Fast Runtime Modules for Probabilistic Networks." *Proceedings 1995 IEEE International Conference on Neural Networks*, (1995).

[172] Rosenblatt, F., *Principles of Neurodynamics.* Spartan Books: East Lansing, MI, (1962).

[173] Rumelhart, D.E., et. al., "Learning internal representations by error propagation." Parallel Distributed Processing: Explorations in *the Microstructures of Cognition*, vol. 1, D.E. Rumelhart, et. al. (Editors), Cambridge, MA, pp. 318-362, (1986).

[174] Schizas, C.N., "Artificial Neural Nets in Computer-Aided Macro Motor Unit Potential Classification." *IEEE Engineering in Medicine and Biology*, pp. 31-38, (1990).

[175] Schizas, C.N., et.al., "Neural Networks : In Search of Computer-Aided Diagnosis." *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 1825-1830, (1991).

[176] Schnupp, P., et. al., *Expert Systems.* Lab Course, Springer-Verlag, (1989).

[177] Schreinemakers, T.F. and Touretzky, D.S., "Interfacing a Neural Network with a Rule-Based Reasoner for Diagnosing Mastitis." *Applications Track of the Proceedings of the International Joint Conference on Neural Networks*, vol. 2., pp. 487-490, (1990).

[178] Shamsolmaah, A., et.al., "A Knowledge-Based System Coupled to a Mathematical Model for Interpretation of Laboratory Data." *IEEE Engineering in Medicine and Biology Magazine*, pp. 40-46, (1993).

[179] Shen, Z., et.al., "Neural Network Approach for Detecting Heart Disease." in *Artificial Neural Networks*, Edited by I. Aleksander and J. Taylor, Elsevier Science Publishers, (1992).

[180] Sheperd, G. M. (Editor), *The Synaptic Organization of the Brain.* Oxford University Press, New York, (1990).

[181] Shepherd, G. M., *Neurobiology*. 2nd ed., Oxford University Press, New York. (1988)

[182] Sherrah J., Jain, R.K., "Classification of Heart Disease Data using the Evolutionary Pre-Processor." *Engineering Mathematics and Application Conference*, Adelaide, pp. 463-466, (1998).

[183] Sherrah, J., *Automatic Feature Extraction for Pattern Recognition* (PhD thesis) The University of Adelaide (1998).

[184] Shiffman, S., et.al., "Building a Speech Interface to a Medical Diagnostic System." *IEEE Expert*, pp. 41-50, (1991).

[185] Shigeo, A., and Lan, M., "A method for Fuzzy Rules Extraction Directly from Numerical Data and Its Application to Pattern Classifiacation." *IEEE Trans. Fuzzy Systems*, vol. 3, No. 1, pp. 18-28, (1995).

[186] Shigeo, A., and Thawonmas, R."A Fuzzy Classifier with Ellipsoidal Regions." *IEEE Trans. Fuzzy Systems*, vol. 5, No. 3, pp. 358-368, (1997).

[187] Shigeo, A., et al.,"Tuning of a Fuzzy Classifier derived from data." *International JOURNAL Approximate Reasoning*, vol. 14, No. 1, pp. 1-24, (1996).

[188] Specht, D. F., "Probabilistic Neural Networks." *Neural Networks*, vol. 3, pp. 109-118, (1990).
[189] Staugaard, Jr., A.C., *Robotics and AI.* Prentice-Hall, U.S.A., (1987).
[190] Takagi, H., and Hayashi, I., "NN Driven Fuzzy Reasoning." *International JOURNAL Approximate Reasoning*, vol. No. 5, no.3, pp. 191-212, (1991).
[191] Taylor, W.A., *What Every Engineer Should Know about Artificial Intelligence.* The MIT Press, England, (1988).
[192] Terano, T. et. al., *Fuzzy Systems Theory and Its Applications.* Academic Press (1992).
[193] Tobi, T. and Anbe, J., "A Practical Application of Fuzzy Theory to an Auto-Regulation System for Extra-Corporeal Circulation." *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks* pp. 905-1001, (1992).
[194] Tom, M. D. and Tenorio, M. F., "A Neural Computation Model with Short-term Memory." *IEEE Trans. Neural Network*, vol. 6, pp. 387-397, (1995).
[195] Tzafestas, S.G. (Editor), *Knowledge-based System Diagnosis, Supervision and Control.* Plenum Press, New York, (1989).
[196] Uckun, S., et. al., "Managing genetic search in job shop scheduling." *IEEE Expert*, pp. 15-24, (1993).
[197] Uebele, V.,et al., "A Neural Network-Based Fuzzy Classifier." *IEEE Trans. Systems,Man,and Cybernetics*,vol. 25, No. 2, pp. 353-361, (1995).
[198] Wan, E. A., "Time Series Prediction by Using a Connectionist Network with Internal Delay Lines." in *Times Series Prediction: Forecasting the Future and Understanding the Past*, A. Weigand and N. Gershenfeld (Editors), pp. 195-218, Addison-Wesley, (1994).
[199] Watanabe, H. et.al., "The Application of a Fuzzy Discrinination Analysis for diagnosis of Valvular Heart Disease." *IEEE Trans. Fuzzy Systems*, vol. 2, No. 4, pp. 267-276, (1994).
[200] Werbos, P., Beyond Regression *: New Tools for Prediction and Analysis in the Behavioral Sciences* (PhD thesis). Cambridge, MA: Harvard U. Committee on Applied Mathematics (1974).
[201] Wettschereck, D. and Dietterich, T., "Improving the Performance of Radial Basis Function networks by Learning Centre Locations." in *Advances in Neural Information Processing Systems* 4, pp. 1133-1140, Morgan Kauffmann, San Mateo CA (1992).
[202] Whitley, D., Starkweather T., and Bogart C., "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity." *Parallel Computing*, vol. 14, pp. 347-361, (1990).
[203] Williams, R.J. and Zipser, D., "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks." *Technical Report ICS Report 8805*, University of California at San Diego, La Jolla, CA 92093, (1988).
[204] Winston, P.H., *Artificial Intelligence.* Addison-Wesley, (1992).
[205] Wolfgram, D.D., et. al., *Expert Systems for the Technical Professional.* John Wiley and Sons, (1987).
[206] Wray, J. and Green, G., "Neural Networks, Approximation Theory, and Finite Precision Computation." *Neural Networks*, vol. 8, pp. 31-37, (1995).
[207] Yamakawa, T., "A fuzzy inference engine in nonlinear analog mode and it's applicationto a fuzzy logic control." *IEEE Trans. Neural Networks*, vol. 4, No. 3, pp. 496-522, (1993).

[208] Yingwei, Lu, Sundararajan, N. and Saratchandran, P., "Adaptive Nonlinear System Identification using Minimal Radial Basis Function Neural Networks." *Proceedings IEEE International Conference on ASSP* (1996).

[209] Yoon, Y., et.al., "A Desktop Neural Network for Dermatology Diagnosis." *Journal of Neural Network Computing*, 43-52 (1989).

[210] Yoshizawa, et.al., "An Automatic Monitoring and Estimation Tool for the Cardiovascular System Under Ventricular Assistance Using Fuzzy Reasoning." *Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks*, 1013-1016, (1992).

[211] Zadeh, L.A., "Fuzzy Logic.", *IEEE Computer*, pp. 83-93, (1988).

[212] Zadeh, L.A., "Outline of a new approach to the analysis of complex systems and decision process." *IEEE Trans. Systems, Man and Cybernatics*, vol. SM-3, No.1, pp. 28-44, (1993).