NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

PATTERN RECOGNITION USING NEURAL NETWORKS

Graduation Project
COM – 400

Student: Asim Nisar

Supervisor: Assoc. Prof. Dr. Adnan Khashman

Nicosia - 2004

ACKNOWLEDGMENTS

First and Foremost I would like to Thank almighty ALLAH for giving me the strength and sincereness during this Project.

My special Thanks to my Supervisor Assoc.Prof. Dr. ADNAN KHASHMAN, for his valuable advice and outmost support in completing this Project.

Second, I would like to thank my family especially my father NISAR AHMED RANA for giving me the chance to complete my academic study and support me during the preparation of this project.

Third, I would also like to thank my real friend Qasim Nisar for his advice, support and for guiding me in the making of this Project.

And finally, thanks to my wife Asma Asim. We were married last year .Her constant love, support, and encouragement made finishing this impossible task possible.

ABSTRACT

Increasing the complexity of the technological processes, the presence of hard formalized and unpredictable information, the uncertainty of environment leads to non-adequate description of these processes by deterministic methods, and so the development of control system with low accuracy. The effective way to solve this problem is the use of artificial intelligence ideas, such as neural networks. Neural networks have been successfully applied in the field of pattern recognition research. In pattern recognition one is interested in techniques to capture the human ability to recognise and classify patterns.

Humans are very good at recognizing and classifying patterns and thereby extracting knowledge from their environment. Autonomous Systems must also be able to recognize and classify objects from input data obtained from their environment.

This Project aims to provide a thorough grounding in the theory and application of pattern recognition, classification, categorization and concept acquisition. Neural networks and graphical models are flexible tools for modeling data, which can be employed, in a principled statistical way, in pattern recognition schemes.

The main purpose to use neural networks graphical models and related methods to analyze and solve real problems. Also, symbolic algorithms are introduced for extracting knowledge from large datasets of patterns (data mining techniques) where it is important to have explicit rules governing pattern recognition. Problems of coping with noisy and/or missing data as well as temporal and sequential patterns are addressed. The obtained results show the efficiency of the application of pattern recognition system.

TABLE OF CONTENTS

INTRODUCTION

Artificial intelligence (AI) is the sub-field of computer science that attempts to develop machines which are capable of emulating human perception. Initial research into AI was mainly directed toward non-numeric computation and symbolic reasoning, because it was realized that these formed the basis of cognitive activities. Various forms of symbolic notation were defined and clever methodologies were devised to assist in reasoning, planning, and learning in problem domains where conventional numerical approaches were deemed inadequate. Although many of these techniques were successfully applied to practical problems in areas such as speech recognition, image recognition and pattern recognition.

Artificial Neural Networks Based on biological theory of human brain, neural networks (NN) are models that attempt to parallel and simulate the functionality and decision-making processes of the human brain. In general, a neural network is referred to as mathematical models of theorized mind and brain activity. Neural network features corresponding to the synapses, neuron, and axons of the brain are input weights. Processing Elements (PE) is the analogs to the human brain's biological neuron. A processing element has many input paths, analogous to brain dendrites. The information transferred along these paths is combined by one of a variety of mathematical functions. The result of these combined inputs is some level internal activity for the receiving PE. The combined input contained within the PE is modified by the transfer function before being passed to other connected PEs whose input paths are usually weighted by the perceived synaptic strength of neural connections. Neural networks have been applied in many applications such as: automotive, aerospace, banking, medical, and robotics.

The objective of this project is to design a neural network for the application of pattern recognition system and investigate the accuracy and time. Also diagnostics the effects of neural networks by recognize the Noisy patterns.

Chapter one describes the introduction of the pattern recognition. And give details about Pattern classification system (supervised and unsupervised). It explained three major approaches of pattern recognition system; Statistical, Syntactic or structural and artificial neural networks.

Chapter two which describes the biological Neural Networks, some explanation of Artificial Neural Networks and Learning Methods. Modification of the weights to the training process and the network subject to this process. A couple of examples of a Hopfield network, one of them for pattern recognition.

Chapter three, is about the application of pattern recognition using neural network back propagation technique which is a supervised learning. It provides the training of back propagation and the problem of the error measure solved in detail. A flowchart of the network and the implementation using the C++ programming language is also explained.

CHAPTER ONE

INTRODUCTION TO PATTERN RECONGITION

1.1 Overview

In what follows this chapter is intended to help the reader understand what the pattern is and discuss a wide range of methods for pattern recognition by neural networks. In general, it will begin with a discussion of underlying theory. This more generic discussion will be followed by specific implementations and practical suggestions. A detailed about pattern recognition systems is provided; definitions, examples and variations found in the literature along with reported results. And last thing the summary of this chapter .

1.2 What is a Pattern?

What is a pattern? A pattern is essentially an arrangement or an ordering in which some organization of underlying structure can be said to exist. We can view the world as made up of patterns. Watanabe (1985) [1] defines a pattern as an entity, vaguely defined, that could be given a name.

A pattern can be referred to as a quantitative or structural description of an object or some other item of interest. A set of patterns that share some common properties can be regarded as a pattern class. The subject matter of pattern recognition by machine deals with techniques for assigning patterns to their respective classes, automatically and with as little human intervention as possible. For example, the machine for automatically sorting mail based on 5-digit zip code at the post office is required to recognize numerals. In this case there are ten pattern classes, one for each of the 10 digits. The function of the zip code recognition machine is to identify geometric patterns (each representing an input digit) as being a member of one of the available pattern classes.

A pattern can be represented by a vector composed of measured stimuli or attributes derived from measured stimuli and their interrelationships. Often a pattern is characterized by the order of elements of which it is made, rather than the intrinsic nature of these elements. Broadly speaking, pattern recognition involves the partitioning or assignment of measurements, stimuli, or input patterns into meaningful categories. It naturally involves extraction of significant attributes of the data from the background of irrelevant details. Speech recognition maps a waveform into words. In character recognition a matrix of pixels (or strokes) is mapped into characters and words. Other examples of pattern recognition

include: signature verification, recognition of faces from a pixel map, and friend-or-foe identification. Likewise, a system that would accept sonar data to determine whether the input was a submarine or a fish would be a pattern recognition system.

1.3 Pattern Recognition Systems

For a typical pattern recognition system the determination of the class is only one of the aspects of the overall task. In general, pattern recognition systems receive data in the form of "raw" measurements which collectively form a stimuli vector. Uncovering relevant attributes in features present within the stimuli vector is typically an essential part of such systems (in some cases this may be all that is required). An ordered collection of such relevant attributes which more faithfully or more clearly represent the underlying structure of the pattern is assembled into a feature vector.

Class is only one of the attributes that may or may not have to be determined depending on the nature of the problem. The attributes may be discrete values, Boolean entities, syntactic labels, or analog values. Learning in this context amounts to the determination of rules of associations between features and attributes of patterns.

Practical image recognition systems generally contain several stages in addition to the recognition engine itself. Before moving on to focus on neural network recognition engines we will briefly describe a somewhat typical recognition system Chen, (1973) [1].

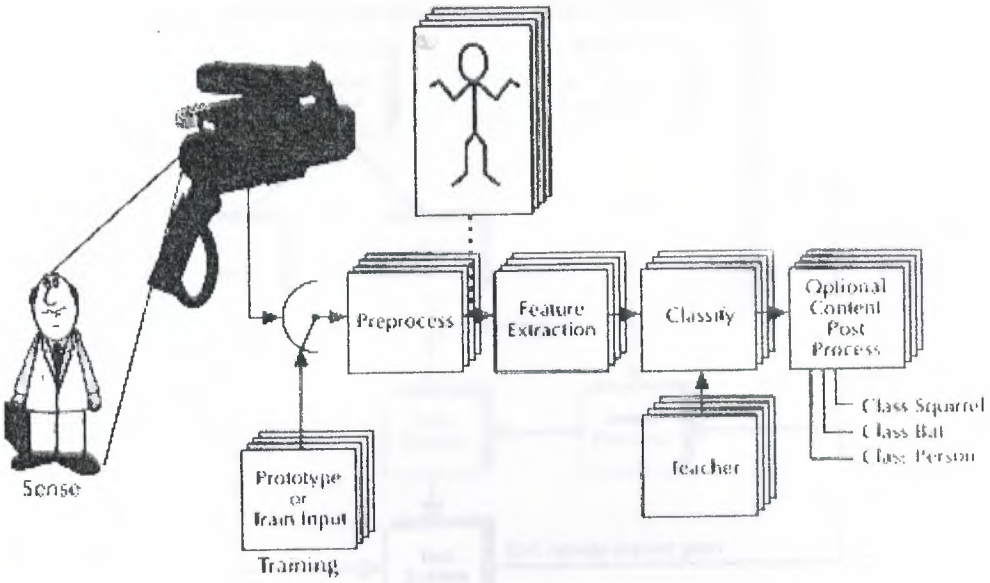

Figure 1.1. Components of a pattern recognition system.

Figure 1.1 shows all the aspects of a typical pattern recognition task:

- *Preprocessing* partitions the image into isolated objects (i.e., characters, etc.). In addition it may scale the image to allow a focus on the object.
- *Feature extraction* abstracts high level information about individual patterns to facilitate recognition.
- The *classifier* identifies the category to which the pattern belongs or, in general, the attributes associated with the given pattern
- The *context processor* increases recognition accuracy by providing relevant information regarding the environment surrounding the object. For example, in the case of character recognition it could be the dictionary and/or language model support.

Figure 1.2 shows the steps involved in the design of a typical pattern recognition system. The choice of adequate sensors, preprocessing techniques, and decision-making algorithm is dictated by the characteristics of the problem domain. Unlike the expert systems, the domain-specific knowledge is implicit in the design and is not represented by a separate module.

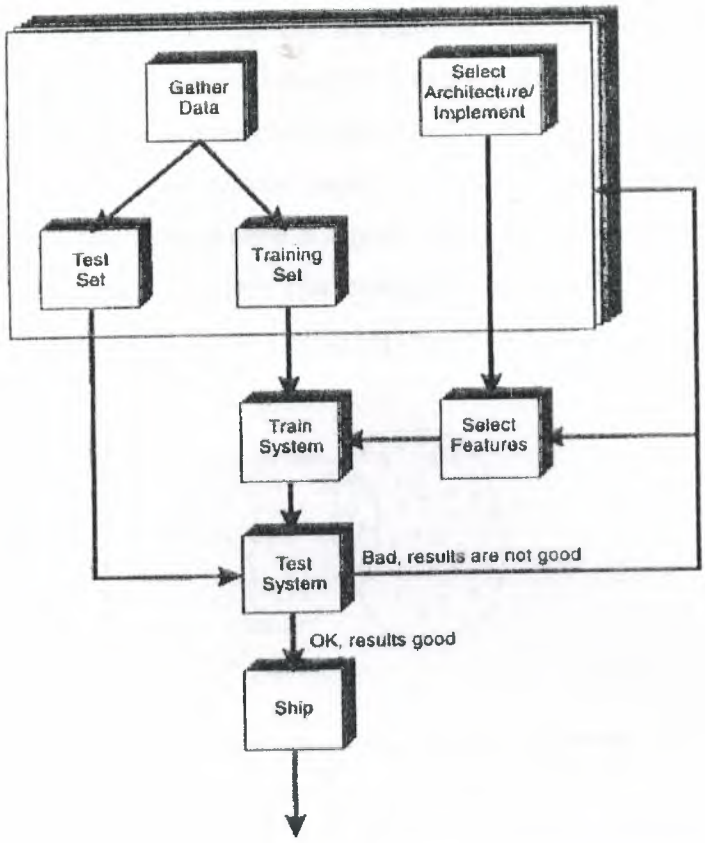

Figure 1.2. A flow chart of the process of designing a learning machine for pattern recognition

A pattern classification system is expected to perform:

(1) Supervised classification, where a given pattern has to be identified as a member of already known or defined classes; or

(2) Unsupervised classification or clustering, where a pattern needs to be assigned to a so far unknown class of patterns.

Pattern recognition may be static or dynamic. In the case of asynchronous systems, the notion of time or sequential order does not play any role. Such a paradigm can be addressed using static pattern recognition, Image labeling/understanding falls into this category. In cases of dynamic pattern recognition, where relative timing is of importance, the temporal correlations between inputs and outputs may a major role. The learning process has to determine the rules governing these temporal correlations. This category includes such applications as control using artificial neural networks or forecasting using neural nets. In the

case of recognizing handwritten characters, for example, the order in which strokes emerge from a digitizing tablet provides much information that is useful in the recognition process.

The task of pattern recognition may be complicated when classes overlap (see Figure 1.3). In this case the recognition system must attempt to minimize the error due to misclassification. The classification error is significantly influenced by the number of samples in the training set. Several researchers (for example, Jain and Chandrasekaran (1982) [2], Fukunaga and Hayes (1989) [3], Foley (1972) [4] have addressed this issue.

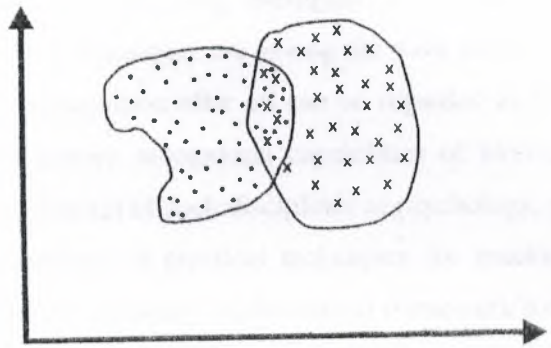

Figure 1.3. Two categories of patterns plotted in the pattern space. Patterns belonging to both classes can be observed in the overlapping region.

The three major approaches for designing a pattern recognition are :

- Statistical
- Syntactic or structural
- Artificial neural networks.

Statistical pattern recognition techniques use the results of statistical communication and estimation theory to obtain a mapping from the representation space to the interpretation space. They rely on the determination of an appropriate combination of feature values that provides measures for discriminating between classes. However, in some cases, the features are not important in themselves. Rather the critical information regarding pattern class, or patterns attributes, is contained in the structural relationships among the features. Applications involving recognition of pictorial patterns (which are characterized by recognizable shapes) such as character recognition, chromosome identification, elementary particle collision photographs, etc. fall into this category. The subject of syntactic pattern recognition deals with this aspect, since it possesses the structure-handling capability lacked

by the statistical pattern recognition approach. Many of the techniques in this field draw from the earlier work in mathematical linguistics and results of research in computer languages.

1.4 Motivation For ANN Approach

The development of a computer as something more than a calculating machine marked the birth of the field of pattern recognition. We have witnessed increased interest in research involving use of machines for performing intelligent tasks normally associated with human behavior. Pattern recognition techniques are among the most important tools used in the field of machine intelligence. Recognition after all can be regarded as a basic attribute of living organisms. The study of pattern recognition capabilities of biological systems (including human beings) falls in the domain of such disciplines as psychology, physiology, biology, and neuroscience. The development of practical techniques for machine implementation of a given recognition task and the necessary mathematical framework for designing such systems lies within the domain of engineering, computer science, and applied mathematics. With the advent of neural network technology a common ground between engineers and students of living systems (psychologists, physiologists, linguists, etc.) was established. We would like to point out that mathematical operations used in theories on pattern recognition and neural networks are often formally similar and identical. Thus, there is good mathematical justification for teaching the two areas together.

Recognizing patterns (and taking action on the basis of the recognition) is the principal activity that all living systems share. Living systems, in general, and human beings, in particular, are the most flexible, efficient, and versatile pattern recognizers known; and their behavior provides ample data for studying the pattern recognition problem. For example, we are able to recognize handwritten characters in a robust manner, despite distortions, omissions, and major variations. The same capabilities can be observed in the context of speech recognition. Humans also have the ability to retrieve information, when only a part of the pattern is presented, based on associated cues. Take, for example, the cocktail party phenomena. At a party you can pick up your name being mentioned in a conversation all the way across the hall even when most of the conversation is inaudible due to a clutter of noise. Similarly, you can recognize a friend in the crowd at a distance even when most of the image is occluded.

Decision-making processes of a human being are often related to the recognition of regularity (patterns). Humans are good at looking for correlations and extracting regularities based on them. Such observations allow humans to act based on anticipation which cuts down the response time and gives an edge over reactionary behavior. Machines are often designed to perform based on reaction to the occurrence of certain events which slows them down in applications such as control.

The nature of patterns to be recognized could be either *sensory recognition* or *conceptual recognition*. The first type involves recognition of concrete entities using sensory information, for example, visual or auditory stimulus. Recognition of physical objects, characters, music, speech, signature, etc. can be regarded as examples of this type of act. On the other hand, conceptual recognition involves acts such as recognition of a solution to a problem or an old argument. It involves recognition of abstract entities and there is no need to resort to an external stimulus in this case. In this book, we shall be concerned with recognition of concrete items only.

The real problem of pattern recognition, however, is to generate a theory that specifies the nature of objects in such a way that a machine will be able to robustly identify them. A study of the way living systems operate provides great insight into addressing this problem. The image in Figure 1.4 indicates the complexity of the type of problem we have been discussing. The image in Figure 1.4(a) shows the face with distinct boundaries between pixels. Thus an image understanding/pattern recognition algorithm, which labels areas with different intensities as parts of different surfaces, would have difficulties in recognizing this pattern of a face. On the other hand, for a human observer it is easier to see that blurring of the boundaries between pixels, as shown in Figure 1.4(b), would result in a easily recognizable face. The ability may be attributed to the existence of interacting high and low spatial frequency channels in the human visual system.

Figure 1.4(a). A facial image with low resolution seen with pixel grid.

One strong objective of the engineering and the artificial intelligence community has been the creation of "intelligent" systems which can exhibit human-like behavior. Such intelligent behavior would enable humans/machine interactions to occur in some fashion that is more natural for the human being. That is, we would like to provide perceptual and cognitive capabilities enabling computers to communicate with us in a fashion that is natural and intuitive to us. One of the goals is to design machines with decision-making capabilities. To accomplish this it is essential that such machines achieve the same pattern information processing capabilities that human beings possess.

Some of the early work in building pattern recognition systems was indeed biologically motivated. The most common historical references are to the devices called perceptron and adaptive linear combiner (ADALINE), respectively. The objective of these studies was to develop a recognition system whose structure and strategy followed the one utilized by humans. Subsequently, with the advent of other, more powerful neural techniques, the field of neural network research is again vigorous. The current serious activity in the area of artificial neural networks and connectionist paradigms is reminiscent of the early period when neurocomputing research flourished.

Figure 1.4(b). Same image blurred to deemphasize the boundaries between pixels.

Some of the early disappointments with the perceptron approach, led some of the researchers to concentrate on the mathematical or computer science aspects of pattern-formatted information processing. For example, emphasis shifted to statistical pattern recognition and classification of patterns with syntactic structures. The neural network, or connectionist paradigm, provides a promising path toward computer systems possessing truly intelligent capabilities. The recent advances in the field of artificial neural networks over the last decade has therefore brought us that much closer to the goal of creating systems exhibiting human-like behavior. Jain and Mao(1994)[5] provide a good discussion on common links between artificial neural network approaches and the statistical pattern recognition approach.

1.5 A Prelude To Pattern Recognition

A pattern can be represented by a set of n numerical measurements or an n-dimensional pattern or measurement vector, **Z**:

$$\mathbf{Z} = z_1, z_2, \ldots, z_{N_m} \qquad (1.1)$$

Subsequently a feature vector, **X**, may be derived from the pattern vector:

$$\mathbf{X} = x_1, x_2, \ldots, x_N \qquad (1.2)$$

Thus a pattern can be viewed as a point in either N_m-dimensional measurement hyperspace or the N-dimensional feature hyperspace. Typically, feature spaces are chosen to be of lower dimensionality than the corresponding measurement space. Pattern classification involves mapping a pattern correctly from the feature/measurement space into a class-membership space. Thus the decision-making process in pattern classification can be summarized as follows. Consider a pattern represented by an n-dimensional feature vector:

$$\mathbf{X} = \left(x_1, x_2, \ldots, x_n\right)^T \qquad (1.3)$$

where T indicates a transpose.

The task is to assign it to one of the K categories, C_1, C_2, . . ., C_K. Note that the measurement vector represents the sensed data, where N_m is the number of measurements. If, for example, an image is represented by an m × m array of pixels with 16 gray levels, we have the dimensionality of the pattern vector, $n = m^2$. Each component $\mathbf{z}_i$ of the vector $\mathbf{Z}$ assumes the appropriate gray level, from the 16 possible values.

Consider the problem of recognizing speech patterns. In this case, the acoustic signals are a function of time. The entities of interest are continuous functions of a variable t, unlike the discrete gray-scale values in the previous example. In order to perform this type of classification we must first measure the observable characteristics of the samples, which involves observing the speech waveform over a period of time in this case. A pattern vector can be formed by sampling these functions at discrete time intervals, t_1, t_2, . . ., t_n, etc. Figure 1.5 shows measurements of time sampled values for a waveform given by $\mathbf{z}(t_1)$, $\mathbf{z}(t_2)$, . . ., $\mathbf{z}(t_n)$.

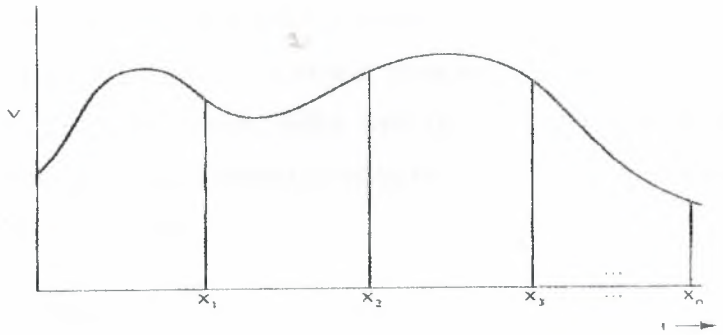

Figure 1.5. Sampling of a waveform at discrete time intervals

A feature vector for speech recognition might, for example, consist of the first N Fourier coefficients of the captured waveform.

Design of a pattern recognition system also involves choosing an appropriate approach into the description of patterns in a form acceptable to the machine in consideration. This decision is also influenced by the nature of the problem domain to which the recognition system will be applied. For example, in the face recognition problem, the image may be converted to an array of pixels with gray-scale representation by means of a photosensitive matrix device (or a camera with a frame grabber). In an application involving color codes, it may be more appropriate to use intensity levels of each of the red, blue, and green (RBG) signals.

Thus, first the feature extractor is designed to find the appropriate features for representing the input patterns, such that the difference between patterns from different classes is enhanced in this feature space. After the feature set is defined and the feature extraction algorithm is in place, a typical recognition process involves two phases: training and prediction. Once the mapping into the feature space has been established, the training phase may begin. Training data that are representative of the problem domain must be obtained. The recognition engine is adjusted such that it maps feature vectors (derived from the training data) into categories with a minimum number of misclassifications. In the second phase (prediction phase), the trained classifier assigns the unknown input pattern to one of the categories/clusters based on the extracted feature vector. The process could be iterative where if prediction results are not acceptable, the choice of features can be revisited or the training can be performed again with different parameters .

Neither raw data representation (bit map or stroke in the case of character recognition) is particularly good for direct input to a neural recognizer. As will be seen, the degree of "badness" will, to a varying extent, differ with the characteristics of the recognizer in question. Some of the problems inherent in using the raw data input formats above as direct inputs to a neural recognizer are

- They are nonorthogonal.
- They are unlikely to represent salient features of the patterns to be recognized.
- They are verbose. Unnecessarily large input vectors lead to a larger than necessary network in which performance during both training and recognition are degraded.
- They are sensitive to slight variations in the image, i.e., font/stroke variations in characters.
- They are likely to contain a good deal of extraneous or nonrelevant information thus providing an invitation to overfitting/overtraining in the recognizer.
- They will not be invariant with respect to translation rotation, etc.

1.6 Statistical Pattern Recognition

In this field the problem of pattern classification is formulated as a statistical decision problem. Statistical pattern recognition is a relatively mature discipline and a number of commercial recognition systems have been designed based on this approach. Pao (1989) [6] is an excellent source of the most relevant techniques from the perspective of practical engineering applications. These present pattern recognition as a problem of estimating density functions in a high-dimensional space and dividing this hyperspace into regions of categories or classes. Decision making in this case is performed using appropriate discriminant functions. Thus mathematical statistics forms the foundation of this subject.

This discipline is also referred to as the decision-theoretical approach since it utilizes decision functions to partition the pattern space. These functions, which are also called discriminant functions, are scalar functions of the pattern **x**. Regions in the pattern space

enclosed by these boundaries provided by the decision functions are labeled as individual classes. A decision function, for n-dimensional pattern space can be expressed as:

$$d_k(\mathbf{x}) = w_k l(\mathbf{x}) \qquad k = 1, 2, \ldots, M \tag{1.4}$$

where w's are coefficients of the decision function corresponding to class C_k and the $l(\mathbf{x})$ are real, single-valued functions of the pattern, $\mathbf{x}$.

The approach is to establish M decision functions $d_1(\mathbf{x})$, $d_2(\mathbf{x})$, . . ., $d_k(\mathbf{x})$, one for each class, such that if a pattern $\mathbf{x}$ belongs to class C_i, then:

$$j = 1, 2, \ldots, M, \quad j \neq i \tag{1.5}$$

Thus we have a relationship that specifies a decision rule. In order to classify a given pattern it is first substituted into all decision functions. Then the pattern is assigned to the class which yields the largest numerical value. Then we have the equation of the decision boundary:

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0 \tag{1.6}$$

which separates classes C_i and C_j. Figure 1.6 shows the block diagram of an automatic classification scheme using discriminant function generators (DFGs).

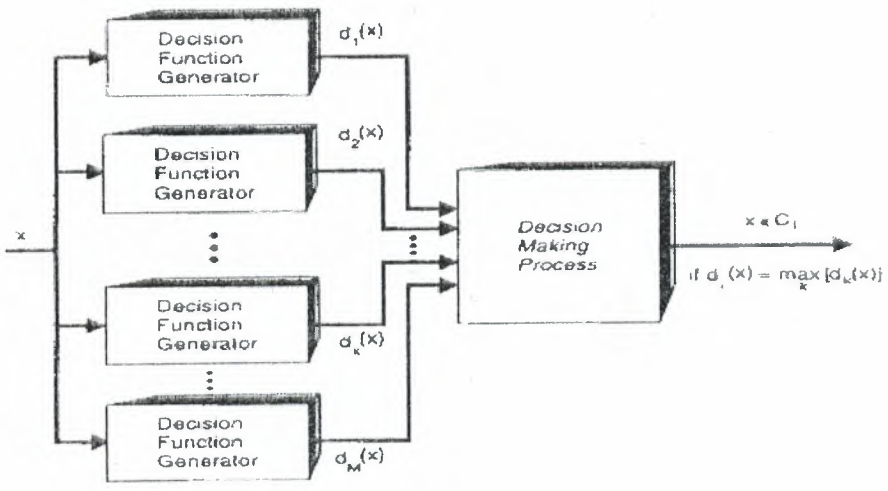

Figure 1.6. Block diagram of a pattern classifier which uses discriminant function generators (DFGs). (Adapted from Tou and Gonzalez (1974) [7]).

Consider a simple example where two measurements are performed on each entity yielding a two-dimensional pattern space which is easy to visualize, for example, the class consisting of professional football players and the class of professional jockeys. Each pattern in this case can be characterized by two measurements: height and weight. Figure 1.7 shows two pattern classes C_1 and C_2 in this two-dimensional pattern space. Thus, $M = 2$, and for all patterns of class C_1:

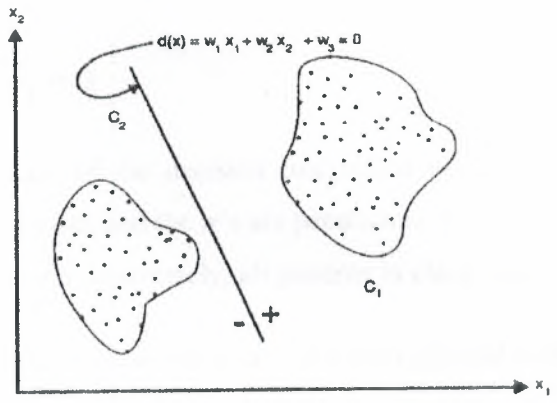

Figure 1.7. Scatter diagram for the feature vectors of two disjoint pattern classes. A simple linear decision function can be used to separate them. (Adapted from Tou and Gonzalez (1974) [7]).

$$d_1(\mathbf{x}) > d_2(\mathbf{x}) \qquad (1.7)$$

and, conversely, for all patterns in class C_2:

$$d_2(\mathbf{x}) > d_1(\mathbf{x}) \qquad (1.8)$$

We can now define:

$$d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) \qquad (1.9)$$

such that it leads to the condition:

$$d(\mathbf{x}) > 0 \quad \text{for} \quad \mathbf{x} \in C_1 \tag{1.10}$$

and

$$d(\mathbf{x}) < 0 \quad \text{for} \quad \mathbf{x} \in C_2 \tag{1.11}$$

In the case of two classes in Figure 1.7 it can be seen that a straight line can separate them. Then we have:

$$d(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 = 0 \tag{1.12}$$

which is a special case of the decision rule stated in equation1.4. Note that (x_1, x_2) represents a pattern in this case, and the w's are parameters. The patterns of class C_2 lie on the negative side of this boundary; conversely, all patterns in class C_1 lie on the positive side.

Note that the decision function in equation 1.4 is quite general in the sense it can represent a variety of complex (including nonlinear) boundaries in n-dimensional pattern space. There are various classification methods that can be used to design a recognition engine for the system. The choice depends on the kind of information that is available about the class-conditional densities. Class-conditional density is the probability function (which estimates the distribution) of pattern $\mathbf{x}$, when $\mathbf{x}$, is from class C_i, and can be given as follows:

$$p(\mathbf{x}/C_i), \quad i = 1, 2, \ldots, K \tag{1.13}$$

If all the class-conditional densities are completely known *a priori*, the decision boundaries between pattern classes can be established using the optimal Bayes decision rule (see Figure 1.8). Since the problem in this case is statistical hypothesis testing, the Bayes classifier gives the smallest error we can achieve from the given distributions. Thus the Bayes classifier is optimal. However, in practical applications the pattern vectors are often of very high dimensionality. This is due to the fact that the number of measurements, n, becomes high in order to ensure that the measurements carry all of the information contained in the original data. In such cases, implementation of the Bayes classifier turns out to be quite difficult due to its complexity.

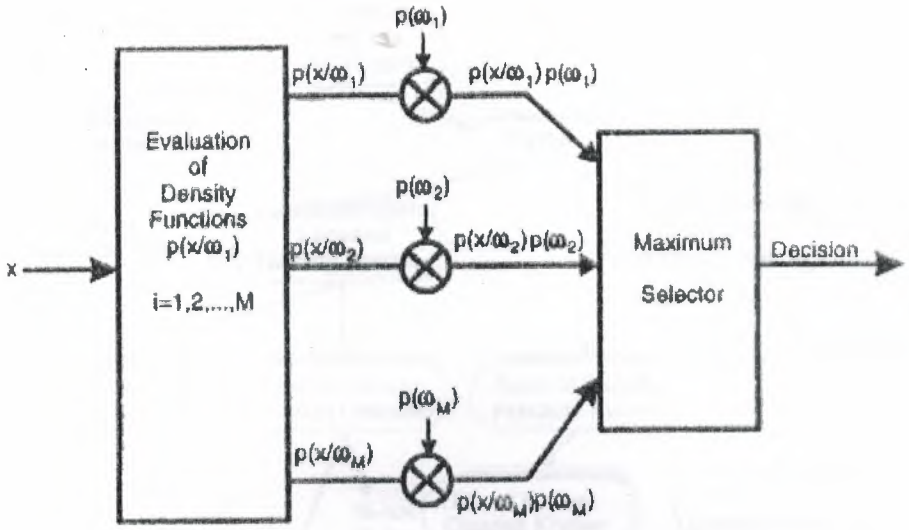

Figure 1.8. Bayes classifier. (Adapted from Tou and Gonzalez (1974) [7]).

Also, in practice, the class-conditional densities are rarely known beforehand and a set of training patterns is needed to determine them. In some cases the functional form of the class-conditional densities is known and the task is to determine the exact values of some of the parameters that are not known. Such a problem is referred to as the parametric decision-making problem. In such cases simpler parametric classifiers are considered. Classifiers with linear, quadratic, and piecewise discriminant functions are the most common choices.

In cases where the precise form of the density function is not known, either it must be estimated or nonparametric methods must be used to obtain a decision rule. pattern recognition as such, statistical techniques do play a role in some neural approaches. The K-nearest-neighbor algorithm, which is of nonparametric category. The Karhunen-Loeve technique is often useful in determining which particular features set is accurate within the degree of tolerance. The radial basis function networks also rely on statistical clustering methods for training the hidden layer neurons. Figure 1.9 shows a tree diagram of various dichotomies which appear in the design of statistical pattern recognition (Jain and Mao, (1994) [5] for details).

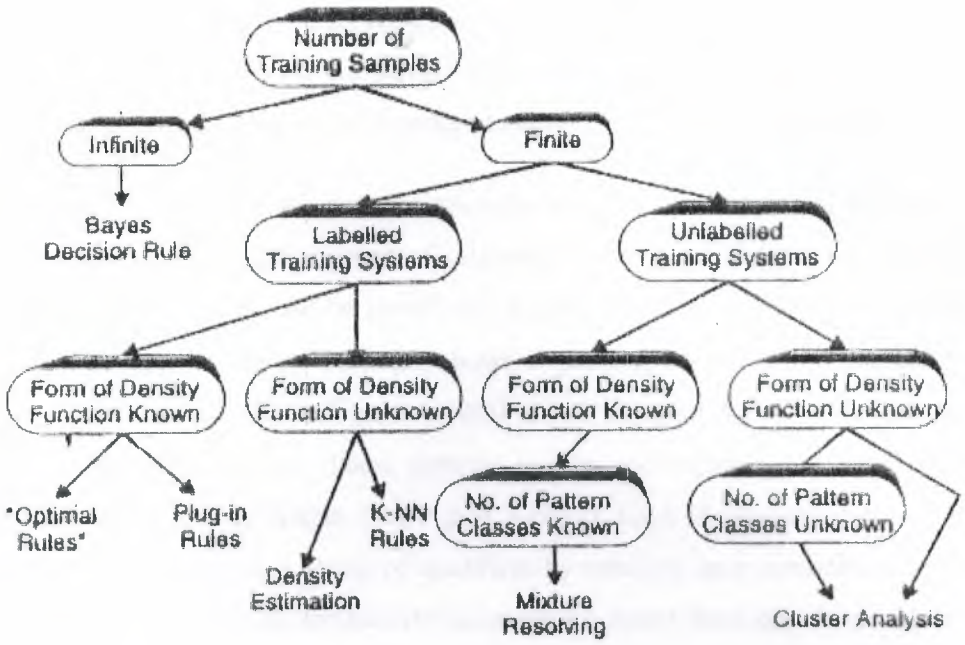

Figure 1.9. Dichotomies in the design of a statistical pattern recognition system. (Adapted from Jain and Mao (1994) [5]).

1.7 Syntactic Pattern Recognition

In applications involving patterns that can be represented meaningfully, using vector notations the statistical pattern recognition approach is ideal. However, this approach lacks a suitable formalism for handling pattern structures and their relationships. For example, in applications like scene analysis, the structure of a pattern plays an important role in the classification process. In this case, a meaningful recognition scheme can be established only if the various components of fundamental importance are identified and their structure, as well as relationships among them, are adequately represented.

In the 1950s several researchers (for example, Chomsky, (1956) [8]) in the field of formal language theory developed mathematical models of grammar. The linguists attempted to apply these mathematical models for describing natural languages, such as English. Once the model is successfully developed it would be possible to provide the computers with the ability to interpret natural languages for the purpose of translation and problem solving. So far these expectations have been unrealized, but such mathematical models of grammar have significantly impacted research in the areas of compiler design, computer languages, and

automata theory. Syntactic pattern recognition is influenced primarily by concepts from formal language theory. Thus, the terms linguistic, grammatical, and structural pattern recognition are also often used in the literature to denote the syntactic approach.

In the syntactic approach the patterns are represented in a hierarchical fashion. That is, patterns are viewed as being composed of subpatterns. These subpatterns may be composed of other subpatterns or they can be primitives. Figures 1.10 (a) and (b) show the different chromosome structures. Figure 1.10 (a) shows a prototype pattern for the class named submedian chromosomes, while Figure 1.10(b) shows the prototype for the second class, called telocentric chromosomes. These patterns can be decomposed in terms of primitives which define various curved shapes (see Figure 1.10[c]). Each chromosome shown in Figure 1.10 can now be encoded as a string of qualifiers by tracking each structure boundary in a clockwise direction. For the submedian chromosome we detect these primitives which can be represented in the form of a string *abcbabdbabcbabdb*. The telocentric chromosome can be represented by the string *ebabcbab*.

We can view the underlying similarities within various structures belonging to the class, submedian chromosomes, as a set of rules of syntax for generation of strings from primitives. A set of rules governing the syntax can be viewed as a *grammar* for the generation of sentences (strings) from the given symbols. Each primitive can be interpreted as a symbol permissible in some grammar. Thus, we can envision two grammars G1 and G2 whose rules allow the generation of strings that correspond to submedian and telocentric chromosomes, respectively. In other words the language L(G1), consisting of sentences (strings) representing submedian chromosomes, can be generated by G1. Similarly, the language L(G2) generated by G2 would consist of sentences representing telocentric chromosomes. Thus, for the determination of the class of the chromosome using the syntactic pattern recognition approach first the two grammars G1 and G2 have to be established. In order to establish a given input pattern (i.e., determine which type of chromosome it is) it is decomposed into a string of primitives. This sentence represents the input pattern and now the problem is to determine the language in which this input pattern represents a valid sentence. In the chromosome identification application if the sentence corresponding to the input pattern belongs to the language L(G1), it is classified as a submedian chromosome. On the other hand, if it belongs to language L(G2), it is classified as telocentric chromosome. If it belongs to both L(G1) and L(G2), it is declared ambiguous. If the sentence representing the input pattern is found to be invalid over both the languages, the input pattern is assigned to a

rejection class consisting of all invalid patterns. Techniques for establishing the class membership of syntactic structures, as well as issues involved in forming grammars, are discussed in Gonzalez and Thomason (1978) [9].

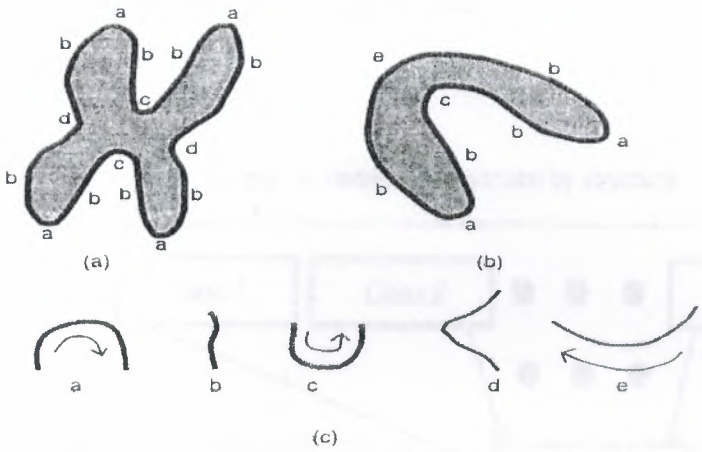

Figure 1.10. A submedian chromosome; (b) a telocentric chromosome; (c) five primitives that can be used to code the two types of chromosomes.

For multiclass pattern recognition problems more grammars (at least one for each class) have to be determined. The pattern is assigned to class *i* if it is a sentence of only L(Gi) and no other language. Thus the syntactic pattern recognition approach in this case is the same as that described for the two-class problem.

The foregoing concepts are valid even in cases where patterns are represented by other data structures instead of strings (i.e., trees and webs (undirected, labeled graphs)).Figure 1.11 shows a typical pattern recognition system designed for classifying patterns using a syntactic approach.

1.8 The Character Recognition Problem

The problem of designing machines that can recognize patterns is highly diverse. It appears in many different forms in a variety of disciplines. The problems range from practical to the profound. The great variety of pattern recognition problems makes it difficult to say what pattern recognition is. However, a good idea of the scope of the field can be given by considering some typical pattern recognition tasks.

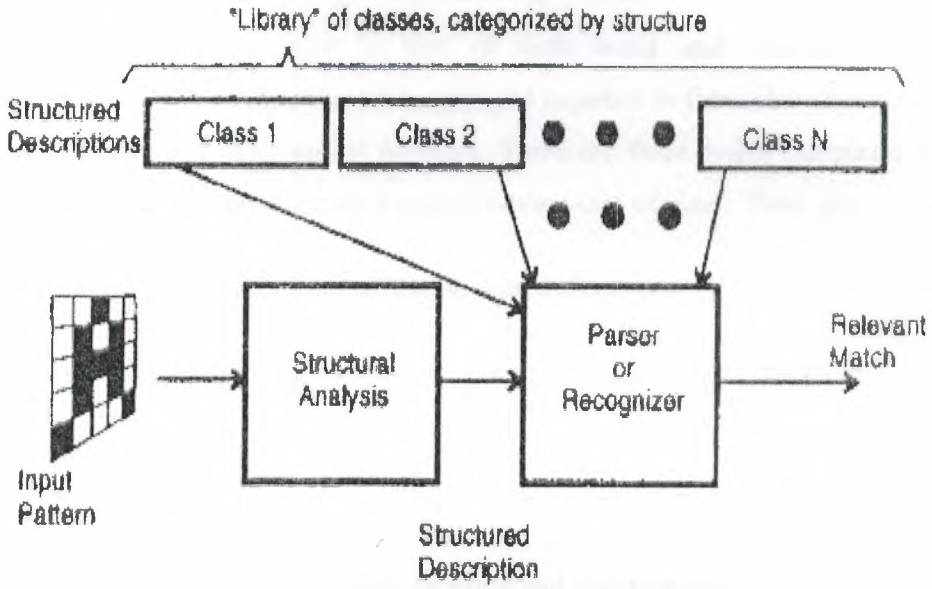

Figure 1.11. Block diagram of a syntactic pattern recognition system for classification.

Frequently a great deal of preprocessing may be required before the "act of recognition" can even begin. We therefore will focus primarily on character recognition since there is an abundance of available data and a minimum of preprocessing. After all, character recognition is one of the classic examples of pattern recognition. There is little loss of generality in that the fundamental neural recognition techniques will be similar to those used in other problem domains. Furthermore, our objective of taking a hands-on practitioners approach is facilitated by using character recognition as an exemplar problem.

The character recognition problem is widely studied in the pattern recognition literature, yet is far from being a solved problem. Nevertheless, it is tractable in the sense that a great amount of data can be easily obtained. One of the most common divisions between character recognition systems lies in whether the recognizer is focused on handwritten text or machine printed text.

Handwritten text data presented to the system may be either on-line or off-line. On-line handwritten text input from a tablet is presented as a sequence of coordinates $v(x, y, t)$ where t is time. Stroke order is available in this context as an aid to recognition. The down side is that handwritten text is significantly less constrained than printed text. Another issue that arises in the context of handwritten text is that of both word and character segmentation. Determination of which strokes should be grouped together to form characters and of where word boundaries exist is a nontrivial problem. There are three major categories into which noncursive text may be grouped from a segmentation point of view. These are

- Box mode — Characters are written in a predefined box.
- Ruled mode — Characters (and words) are written on a predefined line.
- Unruled mode — Characters (and words) may be written anywhere on the input surface and may also slope arbitrarily.

In box mode, segmentation is trivial. In ruled and unruled mode, segmentation problems could turn out to be very difficult. The crucial importance of segmentation should not be underestimated. Incorrect segmentation can and will lead to poor recognition by the overall system. These and other issues render the recognition of handwritten characters more formidable than the machine printed character recognition problem even where the goal is omnifont recognition.

In the case of optical character recognition (OCR) (which can also be regarded as off-line), printed or handwritten text will be represented by a bit-mapped image typically from a scanner. Segmentation is less of a problem in this context although a preprocessing stage is still required. Even though printed text is more constrained than handwritten, the recognition of machine printed text remains challenging. Figure 1.12 below illustrates character confusion in machine printed text recognition.

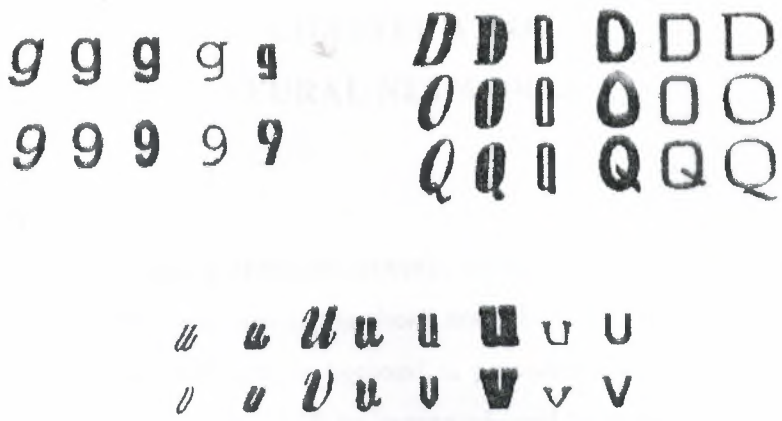

Figure 1.12. Some machine printed text with noisy characters.

In what follows we will not make a great deal of distinction between handwritten as opposed to printed characters in that there will be no attempt to deal with using stroke order information. In this sense we deal with handwritten characters (as seems to be the case in most of the literature) as though they were a kind of very highly unconstrained printed text.

1.9 Summary

The chapter presented the introduction of pattern recognition, and provides the different definitions and examples about pattern and pattern recognition systems. It shows all aspects of a typical pattern recognition, components of pattern recognition system. And gives details about Pattern classification system (supervised and unsupervised). It explained three major approaches of pattern recognition system; Statistical, Syntactic or structural, and Artificial neural networks.

CHAPTER TWO
NEURAL NETWORKS

2.1 Overview

This chapter is an overview of neural networks which helps the reader to understand what Artificial Neural Networks are, how to use them, and where they are currently being used. A detailed biological neural network background is provided; definitions and the biological nervous system, such as the brain will be presented, and how the artificial neurons work. Also we will see the different between the neural network computing and both of the tradition computing and expert computing. Advantages and disadvantages of neural networks. And last thing the summary of this chapter.

2.2 Biological Neural Networks

The current view of the nervous system owes much to the two pioneers, Ramony Cajal (1934) and Sherrington (1933) [10] who introduced the notion that the brain is composed of distinct cells (neurons). The brain has approximately 100 billion (10^{11}) nerve cells (neurons) and it is estimated that there are about 100 trillion (10^{14}) connections (synapses) having a density of 1000 connections per neuron. As a result of this truly staggering number of neurons and synapses, the brain is an enormously efficient structure, even though neurons are much slower computing elements compared with the silicon logic gates. In a silicon chip, events happen at the rate of few nanoseconds (10^{-9} seconds), while neural events occur at the rate of milliseconds (10^{-3} seconds).

The nervous system of humans and other primates consists of three stages as shown in Figure 2.1. The sensory stimuli from the environment or human body is converted into electrical impulses by the receptors, such as eyes, ears, nose, skin, etc. These information-bearing signals are then passed on through forward links to the brain which is central to the nervous system. The brain in Figure 2.1 is represented by a neural network.

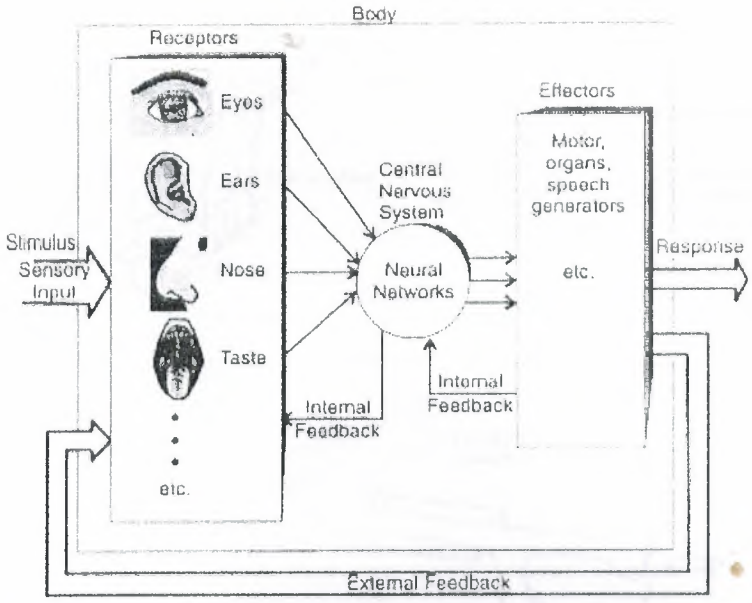

Figure 2.1. Block diagram of the nervous system showing the information flow through the links.

The brain continually receives information which it processes, evaluates, and compares to the stored information and makes appropriate decisions. The necessary commands are then generated and transmitted to the effectors (motor organs like tongue, vocal cords, etc. for speech) through forward links. The effectors convert these electrical impulses into discernible responses as system outputs. At the same time motor organs are monitored in the central nervous system by feedback links that verify the action. The implementation of these commands function through both external and internal feedback for acts, such as hand-eye coordination. Thus, the overall system bears some resemblance to a closed-loop control system.

Figure 2.2 shows the schematic diagram of a "generic neuron" with its main components labeled as axon, cell body (soma), dendrites, and synapses. Figure 2.2 also shows the characteristic ions (Na^+ K^+, and Cl^-) where they are prevalent inside and outside the cell membrane.

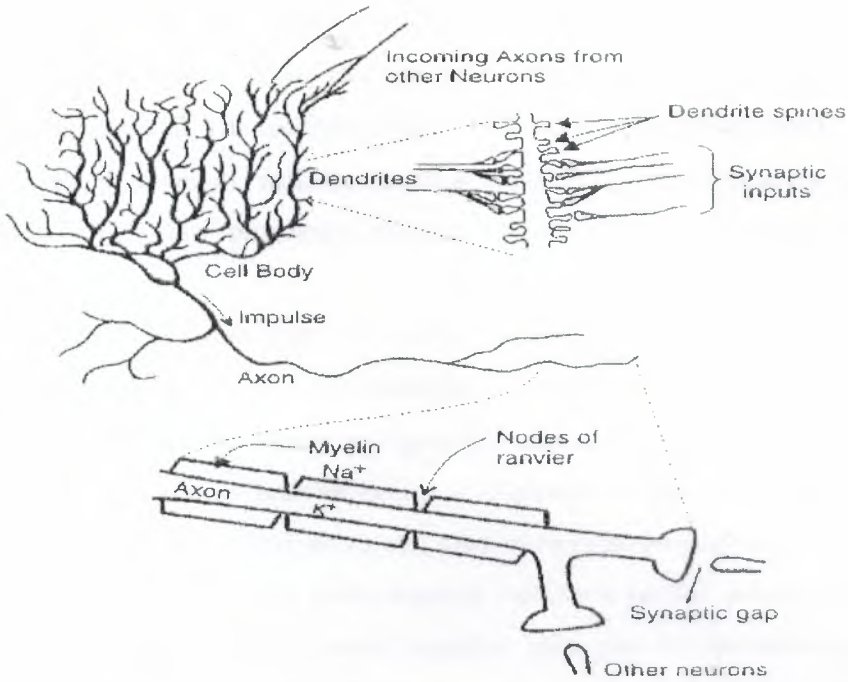

Figure 2.2. A neuron with its components labeled.

Dendrites (with many small branches resembling a tree) are the receptors of electrical signals from other cells. Axons, the transmission lines, carry the signals away from the neuron. They have a smoother surface, fewer branches, and greater length compared with dendrites which have an irregular surface. The soma (cell body) contains the cell nucleus (the carrier of the genetic material) and is responsible for providing the necessary support functions to the entire neuron. These support functions include energy generation, protein synthesis, etc. The soma acts as an information processor by summing the electrical potentials from many dendrites.

The interactions between the neurons are mediated through elementary structural and functional units, called synapses. The synapses could be electrical, where the action potential (shown in Figure 2.2) travels between cells by direct electrical condition. However, chemical synapses, where conduction (information transfer) is mediated by a chemical transmitter, are more common. The synapse can impose excitation or inhibition on the receptive neuron.
The chemical synapse operates as follows :

- The transmitting neuron, called presynaptic cell, liberates a transmitter substance that diffuses across the synaptic junction. Thus an electrical signal is converted to a chemical signal.

- The chemical neurotransmitter causes a positive increase (for an excitatory connection) and a decrease (for an inhibitory connection) in the postsynaptic membrane potential. The receiving neuron is called the postsynaptic cell.
- Thus, at the postsynaptic cell, the chemical signal is converted back into an electrical potential which now propagates through to the other components of the neural network.

In various parts of the brain there are a wide variety of neurons, each with a different shape and size. Also, the number of different types of synaptic junctions between the cells is quite large. The cell membrane, shown in Figure 2.1, consists of myelin sheaths (electrically insulating layer) with nodes of Ranvier acting as channels for ion transfer. It plays a very important role in the activities of the nerve cell, such as impulse propagation.

Figure 2.3 (a) shows a trace of the nerve impulse waveform (action potential) as it would appear on an oscilloscope. Such a nerve impulse train can be recorded by placing a microelectrode near an axon. Figure 2.3 (b) shows the corresponding nerve impulse train. Long-term memories are thought to be defined in the nervous system in terms of variations in synaptic strengths. The changes in synaptic efficiency are mediated through biochemical changes associated with learning and memory. Experimental evidences supporting this universal assumption are as follows:

- Changes in strength in specific synapses in hippocampal neurons depend on combined activity of multiple inputs;
- Changes in the morphology of dendritic spines contribute substantially to learning and memory in central neurons;
- Calcium ions mediate changes in synaptic efficiency contributing to increases post-synaptic receptors, proteins synthesis involved in spine swelling, transport of dendritic microtubules, and output of pre-synaptic transmitter.

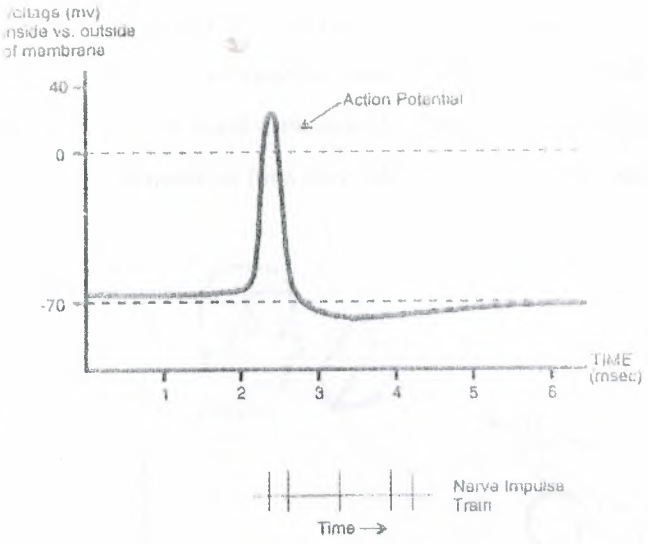

Figure 2.3. (a) Trace of a nerve impulse waveform;
(b) corresponding nerve impulse train.

2.3 Hierarchical Organization in the Brain

Extensive research on the analysis of local regions in the brain Churchland and Sejnowski,(1992) [11] has revealed the structural organization of the brain with different functions taking place at higher and lower levels in the brain. Figure 2.4 shows the hierarchy of the levels of organization in the brain. The traditional digital computers can also be viewed as a structured system, but the organization is radically different. Thus, a look into the levels of organization of the nervous system may provide new insights into designing computers with a radically different organization.

At the top the behavior of an individual is determined at the "whole brain" level. The behavior is mediated by topographic maps, systems and pathways at the inter-regional circuit level beneath it. Topographic maps involve multiple regions located in the different parts of the brain and they are organized to respond to the sensory information. In fact the visual system, the motor system, and the auditory system taken as a whole fit into this category. The third level of complexity is called local circuitry and is made up of neurons with similar or different properties. These neuronal assemblies are responsible for local processing. The next level is the neuron itself, about 100 micrometer in size, containing several *dendric subunits*. Below this level lies the *neural microstructure* (like a silicon chip made up of an assembly of transistors in the case of a computer) which produces various functional operations. These are structures that affect areas around the synapses and are of the size of a few microns, with a speed of operation of a few milliseconds (quite slow compared with a speed of nanoseconds

for transistors in traditional computers). The fact that neurons are such slow, millisecond devices may partially account for the massive parallelism required for a large biological computer (i.e., the brain). The next level consists of synaptic junctions where cells transmit signals from one to another. Synapses in turn rely on the actions of the molecules and ions at the level below

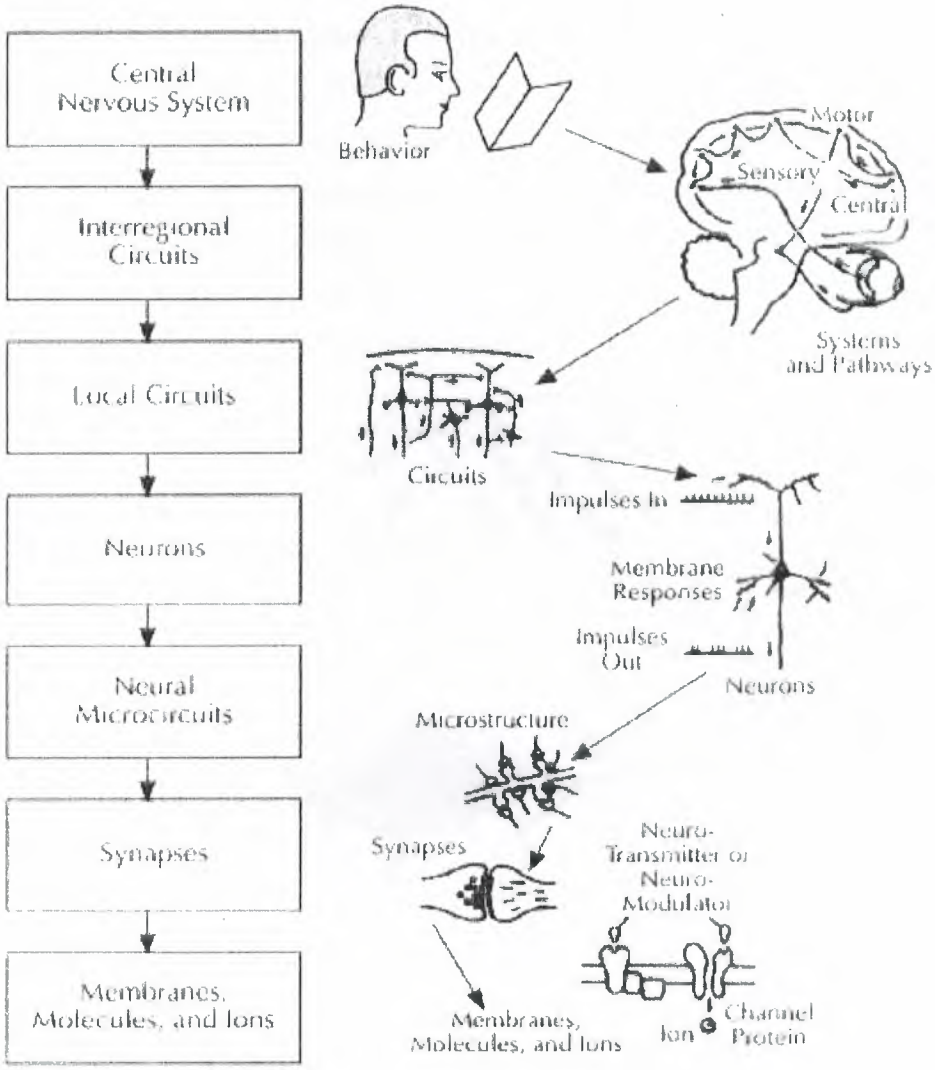

Figure 2.4. Structural organization of levels in biological nervous systems

Neuroscientists have identified different regions of the brain in terms of their specialization for complex tasks, such as vision, speech, hearing, etc. The visual information is analyzed in the back side of the brain (in the occipital lobe). The auditory sensors send information to the auditory cortex (upper part of the temporal lobe) where they are analyzed.

The visual cortex in fact contains a map that reflects the layout of the surface of the retina. The cochlea is the part of the inner ear that receives auditory input and the map on the auditory cortex reflects the sheets of receptors in the cochlea. The parietal lobe articipates in processing information from the skin and body. The association cortex carries out higher brain functions like cognition, perception, etc.

The major neural structures within or below the cerebral cortex are shown in Figure 2.5. The cerebellum, at the bottom, is involved in muscular activities such as walking, jumping, playing a musical instrument, etc., which require sensory-motor coordination. Next to it the brainstem is concerned with respiration, heart rhythm, and gastrointestinal functions. The spinal cord, below the brain, transmits signals to and from the brain and generates appropriate reflex actions. The hippocampus in primitive animals participates in finding appropriate responses to various smells in the environment, but in humans it takes on new roles.

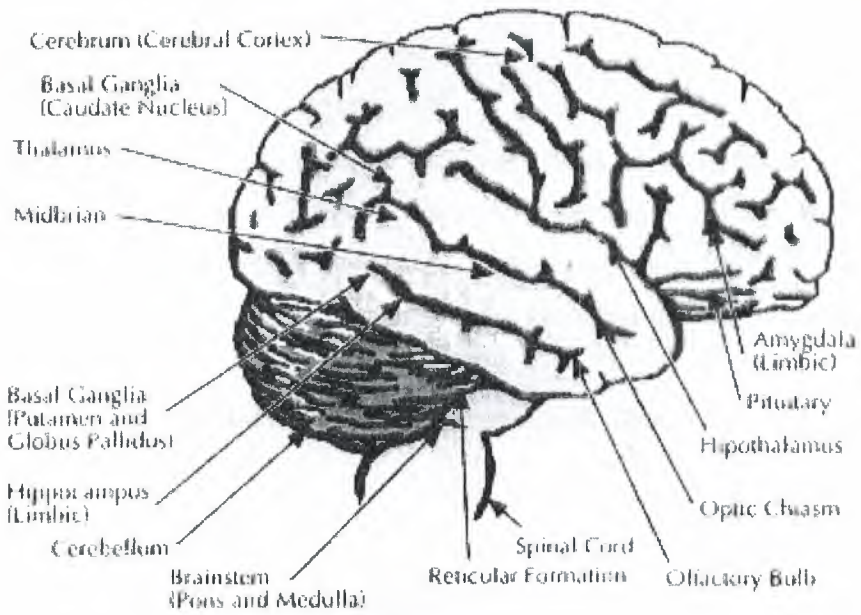

Figure 2.5. Major neural structures in the brain

In the case of the visual system, the retina is the sensory organ that acts as a transducer. This transducer converts the photons (stimulus energy) into corresponding neural signals which are subsequently processed in the brain by the visual cortex. The retina which senses the stimulus for the visual system. The retina itself has five layers, with receptor cells (consisting of rods and cones) receiving light signals from outside. These signals are then transmitted through different layers of cells where some horizontal preprocessing occurs. Finally the ganglion cells transmit the signals to the primary visual cortex, such that they encode the local areas of the visual stimulus. shows the visual pathways, originating from the

retina, via the lateral geniculate nucleus (LGN), to the primary visual cortex. The brain makes a map of the visual field, called topographic maps, at the visual cortex, showed that various neurons in the cat's visual system respond selectively to borders, orientation, motion, length of line, etc. Similarly, in the auditory system the cochlea (the sensory organ) converts the sound waves (stimulus) into neural signals which are subsequently processed by the auditory cortex.

In fact, a tremendous amount of data exists regarding the anatomy of the brain. The locations and functions of various major structures within the nervous system are very well understood. However, the precise conclusions about the role of each part of the neural circuitry are lacking. Neural network models are likely to contribute toward gaining a better understanding of mechanisms and circuitry involved in various functions carried out by the brain.

2.4 Neural networks

Neural networks today began with the pioneering work of McCulloch and Pitts (1943) [12] and has its roots in a rich interdisciplinary history dating from the early 1940s. McCulloch was trained as a psychiatrist and neuroanatomist, while Pitts was a mathematical prodigy. Their classical study of all-or-none neurons described the logical calculus of neural networks.

Figure 2.6 shows a McCulloch-Pitts model of a neuron with inputs x_i, for $i = 1, 2, \ldots, N$.

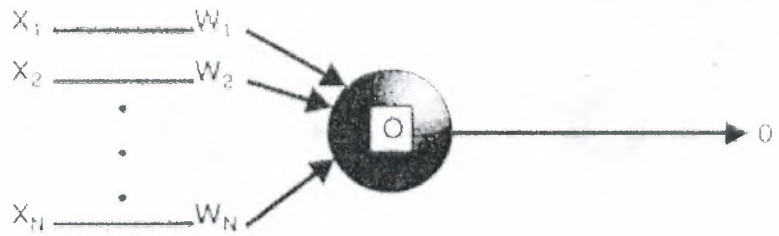

Figure 2.6. A McCulloch-Pitts model of neuron.

W_i denotes the multiplicative weight (synaptic strength) connecting the i^{th} input to the neuron. Theta is the neuron threshold value, which needs to be exceeded by the weighted sum of inputs for the neuron to fire (output = O is the output of the neuron. The weight, W_i, is positive if the connection (synapse) is excitatory and negative if the connection is inhibitory.

The inputs, x_i are binary (0 or 1) and can be from sensors directly or from other neurons. The following relationship defines the firing rule for the neuron:

$$O = g\left(\sum_{i=1}^{N} W_i x_i\right) \tag{2.1}$$

where $g(x)$ is the activation function defined as:

$$g(x) = \begin{cases} 1 \text{ if } x \geq \theta \\ 0 \text{ if } x < \theta \end{cases} \tag{2.2}$$

This simplistic model could demonstrate substantial computing potential, since by appropriate choice of weights it can perform logic operations such as AND, OR, NOT, etc. Figure 2.7 shows the appropriate weights for performing each of these operations. As we know, any multivariate combinatorial function can be performed using either the NOT and AND gates, or the NOT and OR gates. If we assume that a unit delay exists between input and output of a McCulloch-Pitts neuron (as shown in Figure 2.6), we can indeed build sequential logic circuitry with it. Figure 2.8 shows an implementation of a single memory cell that can retain the input.

As seen from the figure, an input of 1 at x_1 sets the output ($O = 1$) while the input of 1 at x_2 resets it ($O = 0$). Due to the feedback loop the output will be sustained in the absence of inputs.

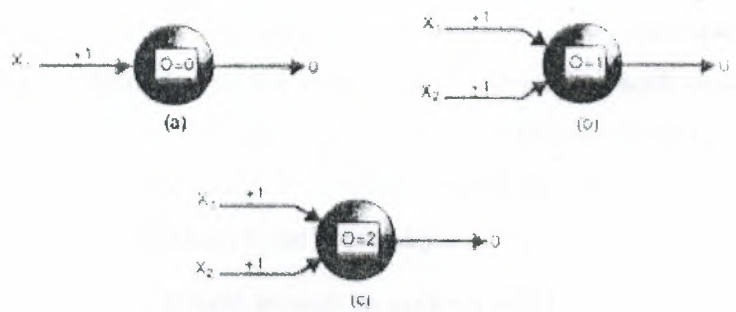

Figure 2.7. (a) Implementation of a NOT gate; (b) an OR gate; and (c) an AND gate implementation

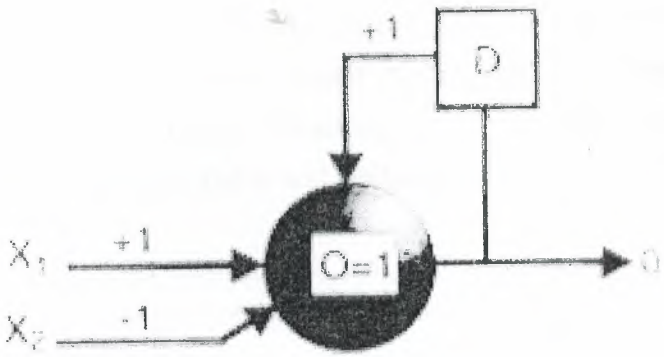

Figure 2.8. Implementation of a memory cell by using a feedback and assuming a delay of one unit of time.

This led to the computer-brain analogy, called cybernetics, based on the fact that neurons are binary, just like switches in a digital computer. Wiener (1948)[13] described important concepts of control, communications, and signal processing based on his perception of similarities between computers and brains, which spurred interest in developing the science of cybernetics. He discussed the significance of statistical mechanics in the context of learning systems, but it was Hopfield (1982, 1984)[14], who established the real linkage between statistical mechanics and neural assemblies. Von Neumann used the idealized switch-delay elements derived from the neuronal models of McCulloch and Pitts to construct the EDVAC computer. He in fact suggested that research in using "brain language" to design brain-like processing machines might be interesting (von Neumann, 1958)[15]. The next major development came when a psychologist, Hebb (1949)[16] proposed a learning scheme for updating the synaptic strengths between the neurons. He proposed that as the biological organisms learn different functional tasks, the connectivity in the brain continually changes. He was also first to propose that neural assemblies are created by such changes. His famous postulate of learning, which we now refer to as the Hebbian learning rule, stated that information can be stored in synaptic connections and the strength of a synapse would increase by the repeated activation of one neuron by the other one across that synapse.

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased."

This learning rule, called the Hebb rule or correlation learning rule, has had a profound impact on the future developments in the field of computational models of learning and

adaptive systems. The original Hebb rule did not contain a provision for selectively weakening (or eliminating) a synapse. Rochester et al.(1956)[17] performed simulations on digital computers to test Hebb's theory of learning in the brain, on an assembly of neurons. They demonstrated that it was essential to add inhibition for the theory to actually work for a neuronal assembly.

Figure 2.9 shows a simple perceptron with sensory elements (S elements), association units (A units), and response units (R units). The sensors could be photoreceptive devices for optical patterns in analogy to the retina where the light impinges.Several S elements, which respond in all-or-none fashion, are connected to each A unit in the association area through fixed excitatory or inhibitory connections. A units in turn are connected to R units in the response area through modifiable connections.

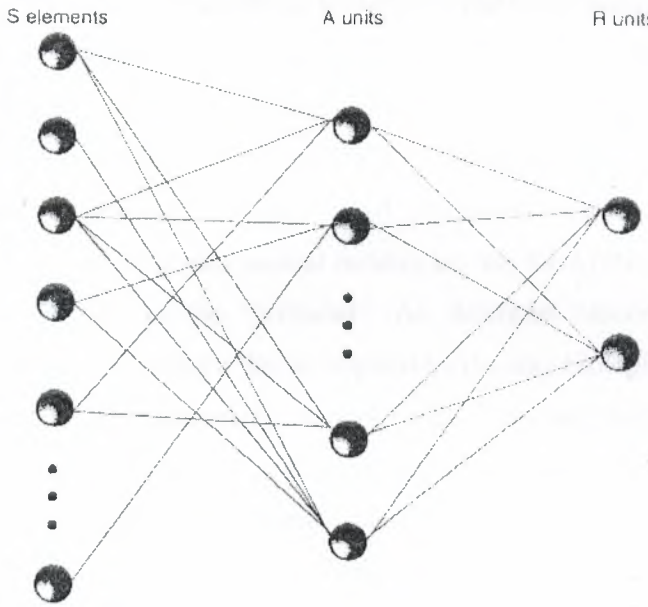

Figure 2.9. A simple perceptron structure with connections between units in three different areas

A perceptron with a single R unit can perform classification when only two classes are involved. For classification involving more than two categories, several R units are required in the response layer. The learning procedure (algorithm) for adjusting the free parameters in

the network shown in Figure 2.9. The proof of convergence of the algorithm, known as the perceptron convergence algorithm, states that if the parameters used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. A learning mechanism where the summed square error in the network output was minimized. So introduced a device called ADALINE (for adaptive linear combiner) based on this powerful learning rule. During the 1960s ADALINE and its extensions to MADALINE (for many ADALINEs) were used in several pattern recognition and adaptive control applications. In the communications industry they were applied as adaptive filters for echo suppression in long-distance telephone communication.

2.5 Artificial Neural Networks

In its most general form a network of artificial neurons, as information processing units, is inspired by the way in which the brain performs a particular task or function of interest. Or neural network in a broader sense such that the neural nets of the actual brain are included in the field of study and provide room for a consideration of biological findings. when we are talking about a neural network, we should more properly say "artificial neural network" (ANN), because that is what we mean most of the time. Biological neural networks are much more complicated than the mathematical models we use for ANNs. But it is customary to be lazy and drop the "A" or the "artificial". An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific Problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

- **Definition:**

A neural network is a massively parallel-distributed processor made up of simple processing units, which has neural propensity for storing experiential knowledge making it available for use. It resembles the brain in two respects:

1.The network from its environment through a learning process acquires knowledge.
2.Interneuron connection strength, known as synaptic weights, is used to store the acquired knowledge.

A neural network is a massively parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use.

It resembles the brain in two respects:

1.Knowledge is acquired by the network through a learning process.
2.Interneuron connection strengths known as synaptic weights are used to store the knowledge.

2.6 Neural Networks, Traditional Computing and Expert Systems

Neural networks offer a different way to analyze data, and to recognize patterns within that data, than traditional computing methods. However, they are not a solution for all computing problems. Traditional computing methods work well for problems that can be well characterized. Balancing checkbooks, keeping ledgers, and keeping tabs of inventory are well defined and do not require the special characteristics of neural networks. Table 1.1 identifies the basic differences between the two computing approaches.

- **Traditional computers** are ideal for many applications. They can process data, track inventories, network results, and protect equipment. These applications do not need the special characteristics of neural networks.
- **Expert systems** are an extension of traditional computing and are sometimes called the fifth generation of computing. (First generation computing used switches and wires. The second generation occurred because of the development of the transistor. The third generation involved solid-state technology, the use of integrated circuits, and higher level languages like COBOL, FORTRAN, and "C". End user tools, "code generators," are known as the fourth generation.) The fifth generation involves artificial intelligence.

Table 2.1. Comparison of Computing Approaches.

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing style Functions	Sequential Logically (left brained) via Rules Concepts Calculations Memory and Processing elements separate Cycle time in nanosecond	Parallel Gestault (right brained) via Images Pictures Controls Memory and Processing elements are collected Cycle time in milliSec.
Learning Method Applications	By rules (didactically) Accounting Word Processing Math inventory Digital Communications	By example (Socratically) Sensor Processing Speech Recognition Pattern Recognition Text Recognition

Typically, an expert system consists of two parts, an inference engine and a knowledge base. The inference engine is generic. It handles the user interface, external files, program access, and scheduling. The knowledge base contains the information that is specific to a particular problem. This knowledge base allows an expert to define the rules which govern a process. This expert does not have to understand traditional programming. That person simply has to understand both what he wants a computer to do and how the mechanism of the expert system shell works. It is this shell, part of the inference engine that actually tells the computer how to implement the expert's desires. This implementation occurs by the expert system generating the computer's programming itself; it does that through "programming" of its own. This programming is needed to establish the rules for a particular application. This method of establishing rules is also complex and does require a detail oriented person. Efforts to make expert systems general have run into a number of problems. As the complexity of the system increases, the system simply demands too much computing resources and becomes too slow. Expert systems have been found to be feasible only when narrowly confined.

Artificial neural networks offer a completely different approach to problem solving and they are sometimes called the sixth generation of computing. They try to provide a tool that

both programs itself and learns on its own. Neural networks are structured to provide the capability to solve problems without the benefits of an expert and without the need of programming. They can seek patterns in data that no one knows are there.

2.7 Structure and processing of Artificial Neurons

A neural network is a computational structure inspired by the study of biological neural processing. There are many different types of neural networks, from relatively simple to very complex, just as there are many theories on how biological neural processing works. We will begin with a discussion of a layered feed-forward type of neural network.

A layered feed-forward neural network has layers, or subgroups of processing elements. A layer of processing elements makes independent computations on data that it receives and passes the results to another layer. The next layer may in turn make its independent computations and pass on the results to yet another layer. Finally, a subgroup of one or more processing elements determines the output from the network. Each processing element makes its computation based upon a weighted sum of its inputs. The first layer is the input layer and the last the output layer. The layers that are placed between the first and the last layers are the hidden layers. The processing elements are seen as units that are similar to the neurons in a human brain, and hence, they are referred to as cells, neuromimes, or artificial neurons. A threshold function is sometimes used to qualify the output of a neuron in the output layer. Even though our subject matter deals with artificial neurons, we will simply refer to them as neurons. Synapses between neurons are referred to as connections, which are represented by edges of a directed graph in which the nodes are the artificial neurons.

Figure 2.10 is a layered feed-forward neural network. The circular nodes represent neurons. Here there are three layers, an input layer, a hidden layer, and an output layer. The directed graph mentioned shows the connections from nodes from a given layer to other nodes in other layers.

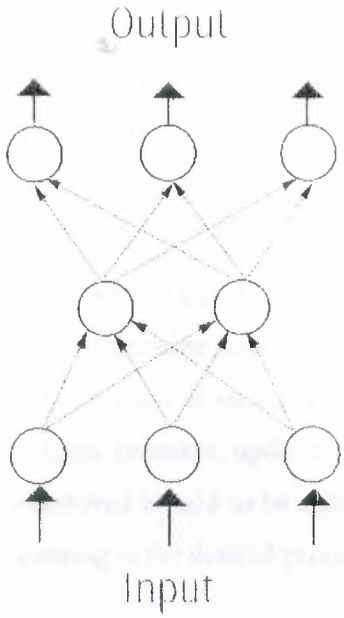

Figure 2.10. A typical neural network

- **Output of a Neuron**

Basically, the internal activation or raw output of a neuron in a neural network is a weighted sum of its inputs, but a threshold function is also used to determine the final value, or the output. When the output is 1, the neuron is said to *fire*, and when it is 0, the neuron is considered not to have fired. When a threshold function is used, different results of activations, all in the same interval of values, can cause the same final output value. This situation helps in the sense that, if precise input causes an activation of 9 and noisy input causes an activation of 10, then the output works out the same as if noise is filtered out.

- **Weights**

The weights used on the connections between different layers have much significance in the working of the neural network and the characterization of a network. The following actions are possible in a neural network:

Start with one set of weights and run the network. (NO TRAINING)

Start with one set of weights, run the network, and modify some or all the weights, and run the network again with the new set of weights. Repeat this process until some predetermined goal is met. (TRAINING)

- **Training**

Since the output(s) may not be what is expected, the weights may need to be altered. Some rule then needs to be used to determine how to alter the weights. There should also be a criterion to specify when the process of successive modification of weights ceases. This process of changing the weights, or rather, updating the weights, is called *training*. A network in which learning is employed is said to be subjected to *training*. Training is an external process or regimen. Learning is the desired process that takes place internal to the network.

- **Feedback**

If you wish to train a network so it can recognize or identify some predetermined patterns, or evaluate some function values for given arguments, it would be important to have information fed back from the output neurons to neurons in some layer before that, to enable further processing and adjustment of weights on the connections. Such feedback can be to the input layer or a layer between the input layer and the output layer, sometimes labeled the hidden layer. What is fed back is usually the error in the output, modified appropriately according to some useful paradigm. The process of feedback continues through the subsequent cycles of operation of the neural network and ceases when the training is completed.

2.8 Supervised or Unsupervised Learning

A network can be subject to supervised or unsupervised learning. The learning would be supervised if external criteria are used and matched by the network output, and if not, the learning is unsupervised. This is one broad way to divide different neural network approaches. Unsupervised approaches are also termed self-organizing. There is more interaction between neurons, typically with feedback and intralayer connections between neurons promoting self-organization.

Supervised networks are a little more straightforward to conceptualize than unsupervised networks. You apply the inputs to the supervised network along with an expected response, much like the Pavlovian conditioned stimulus and response regimen. You mold the network with stimulus-response pairs. A stock market forecaster may present economic data (the stimulus) along with metrics of stock market performance (the response) to the neural network to the present and attempt to predict the future once training is complete.

We provide unsupervised networks with only stimulus. We may, for example, want an unsupervised network to correctly classify parts from a conveyor belt into part numbers, providing an image of each part to do the classification (the stimulus). The unsupervised network in this case would act like a look-up memory that is indexed by its contents, or a Content-Addressable-Memory *(CAM)*.

2.9 Neural Network Construction

There are three aspects to the construction of a neural network:

1. **Structure**—the architecture and topology of the neural network
2. **Encoding**—the method of changing weights
3. **Recall**—the method and capacity to retrieve information

Let's cover the first one—structure. This relates to how many layers the network should contain, and what their functions are, such as for input, for output, or for feature extraction. Structure also encompasses how interconnections are made between neurons in the network, and what their functions are. The second aspect is encoding. Encoding refers to the paradigm used for the determination of and changing of weights on the connections between neurons. In the case of the multilayer feed-forward neural network, you initially can define weights by randomization. Subsequently, in the process of training, you can use the *backpropagation algorithm*, which is a means of updating weights starting from the output backwards. When you have finished training the multilayer feed-forward neural network, you are finished with encoding since weights do not change after training is completed.

Finally, recall is also an important aspect of a neural network. Recall refers to getting an expected output for a given input. If the same input as before is presented to the network, the same corresponding output as before should result. The type of recall can characterize the network as being autoassociative or heteroassociative. Autoassociation is the phenomenon of associating an input vector with itself as the output, whereas heteroassociation is that of recalling a related vector given an input vector. You have a fuzzy remembrance of a phone

number. Luckily, you stored it in an autoassociative neural network. When you apply the fuzzy remembrance, you retrieve the actual phone number. This is a use of autoassociation. Now if you want the individual's name associated with a given phone number, that would require heteroassociation.

The three aspects to the construction of a neural network mentioned above essentially distinguish between different neural networks and are part of their design process.

2.10 Sample Applications

One application for a neural network is pattern classification, or pattern matching. The patterns can be represented by binary digits in the discrete cases, or real numbers representing analog signals in continuous cases. Pattern classification is a form of establishing an autoassociation or heteroassociation. Recall that associating different patterns is building the type of association called heteroassociation. If you input a corrupted or modified pattern A to the neural network, and receive the true pattern A, this is termed autoassociation. What use does this provide? In the human brain example, say you want to recall a face in a crowd and you have a hazy remembrance (input). What you want is the actual image. Autoassociation, then, is useful in recognizing or retrieving patterns with possibly incomplete information as input. What about heteroassociation? Here you associate A with B. Given A, you get B and sometimes vice versa. You could store the face of a person and retrieve it with the person's name, for example. It's quite common in real circumstances to do the opposite, and sometimes not so well. You recall the face of a person, but can't place the name.

2.11 Example—A Feed-Forward Network

A sample feed-forward network, as shown in Figure 2.11, has five neurons arranged in three layers: two neurons (labeled x_1 and x_2) in layer 1, two neurons (labeled x_3 and x_4) in layer 2, and one neuron (labeled x_5) in layer 3. There are arrows connecting the neurons together. This is the direction of information flow. A feed-forward network has information flowing forward only. Each arrow that connects neurons has a weight associated with it (like, w_{31} for example). You calculate the *state,* x, of each neuron by summing the weighted values that flow into a neuron. The state of the neuron is the output value of the neuron and remains the same until the neuron receives new information on its inputs.

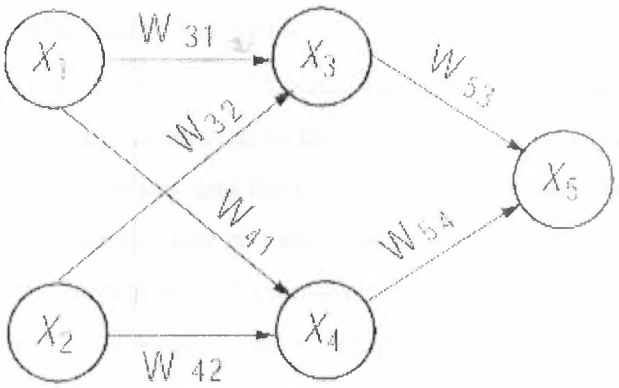

Figure 2.11. A feed-forward neural network with topology 2-2-1.

For example, for $\mathbf{x}_3$ and $\mathbf{x}_5$:

$\mathbf{x}_3 = w_{32}\,\mathbf{x}_2 + w_{31}\,\mathbf{x}_1$

$\mathbf{x}_5 = w_{53}\,\mathbf{x}_3 + w_{54}\,\mathbf{x}_4$

The training algorithms for the feed-forward network called *Backpropagation*. Note that present information to this network at the leftmost nodes (layer 1) called the input layer. we can take information from any other layer in the network, but in most cases do so from the rightmost node(s), which make up the output layer. Weights are usually determined by a supervised training algorithm, where you present examples to the network and adjust weights appropriately to achieve a desired response. Once you have completed training, you can use the network without changing weights, and note the response for inputs that you apply. Note that a detail not yet shown is a nonlinear scaling function that limits the range of the weighted sum. This scaling function has the effect of clipping very large values in positive and negative directions for each neuron so that the cumulative summing that occurs across the network stays within reasonable bounds. Typical real number ranges for neuron inputs and outputs are –1 to +1 or 0 to +1. Now let us contrast this neural network with a completely different type of neural network, the Hopfield network, and present some simple applications for the Hopfield network.

2.12 Example—A Hopfield Network

The neural network we present is a Hopfield network, with a single layer. We place, in this layer, four neurons, each connected to the rest, as shown in Figure 2.12. Some of the connections have a positive weight, and the rest have a negative weight. The network will be presented with two input patterns, one at a time, and it is supposed to recall them. The inputs would be binary patterns having in each component a 0 or 1. If two patterns of equal length are given and are treated as vectors, their dot product is obtained by first multiplying corresponding components together and then adding these products. Two vectors are said to be orthogonal, if their dot product is 0. The mathematics involved in computations done for neural networks include matrix multiplication, transpose of a matrix, and transpose of a vector. The inputs (which are stable, stored patterns) to be given should be orthogonal to one another.

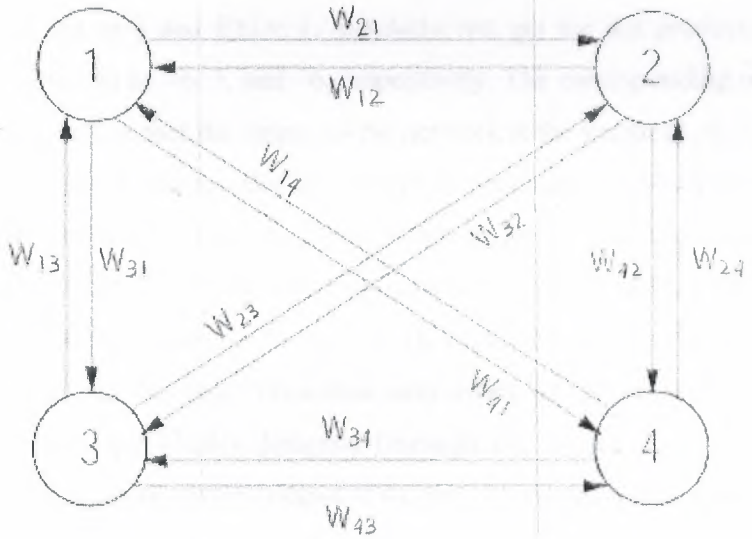

Figure 2.12. Layout of a Hopfield network.

The two patterns we want the network to recall are **A** = (1, 0, 1, 0) and **B** = (0, 1, 0, 1), which you can verify to be orthogonal. Recall that two vectors **A** and **B** are orthogonal if their dot product is equal to zero. This is true in this case since

$$A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 = (1x0 + 0x1 + 1x0 + 0x1) = 0$$

The following matrix W gives the weights on the connections in the network.

$$W = \begin{bmatrix} 0 & -3 & 3 & -3 \\ -3 & 0 & -3 & 3 \\ 3 & -3 & 0 & -3 \\ -3 & 3 & -3 & 0 \end{bmatrix}$$

We need a threshold function also, and we define it as follows. The threshold value [theta] is 0.

$$f(t) = \begin{cases} 1 & \text{if } t >= [\text{theta}] \\ 0 & \text{if } t < [\text{theta}] \end{cases}$$

We have four neurons in the only layer in this network. We need to compute the activation of each neuron as the weighted sum of its inputs. The activation at the first node is the dot product of the input vector and the first column of the weight matrix (0 -3 3 -3). We get the activation at the other nodes similarly. The output of a neuron is then calculated by evaluating the threshold function at the activation of the neuron. So if we present the input vector **A**, the dot product works out to 3 and f(3) = 1. Similarly, we get the dot products of the second, third, and fourth nodes to be –6, 3, and –6, respectively. The corresponding outputs therefore are 0, 1, and 0. This means that the output of the network is the vector (1, 0, 1, 0), same as the input pattern. The network has recalled the pattern as presented, or we can say that pattern **A** is stable, since the output is equal to the input. When **B** is presented, the dot product obtained at the first node is –6 and the output is 0. The outputs for the rest of the nodes taken together with the output of the first node gives (0, 1, 0, 1), which means that the network has stable recall for **B** also. So far we have presented easy cases to the network—vectors that the Hopfield network was specifically designed (through the choice of the weight matrix) to recall. What will the network give as output if we present a pattern different from both **A** and **B**? Let **C** = (0, 1, 0, 0) be presented to the network. The activations would be –3, 0, –3, 3, making the outputs 0, 1, 0, 1, which means that **B** achieves stable recall. This is quite interesting. Suppose we did intend to input **B** and we made a slight error and ended up presenting **C**, instead. The network did what we wanted and recalled **B**. But why not **A**? To answer this, let us ask is **C** closer to **A** or **B**? How do we compare? We use the distance formula for two four-dimensional points. If (a, b, c, d) and (e, f, g, h) are two four-dimensional points, the distance between them is:

[radic][(a – e)2 + (b – f)2 + (c – g)2 + (d – h)2]

The distance between **A** and **C** is [radic]3, whereas the distance between **B** and **C** is just 1. So since **B** is closer in this sense, **B** was recalled rather than **A**. verify that if we do the same exercise with **D** = (0, 0, 1, 0), note that the network recalls **A**, which is closer than **B** to **D**.

2.13 Example-Hopfield Network through a bipolar vector

An application of Kohonen's feature map for pattern recognition. Here we give an example of pattern association using a Hopfield network. The patterns are some characters. A pattern representing a character becomes an input to a Hopfield network through a bipolar vector. This bipolar vector is generated from the pixel (picture element) grid for the character, with an assignment of a 1 to a black pixel and a -1 to a pixel that is white. A grid size such as 5x7 or higher is usually employed in these approaches. The number of pixels involved will then be 35 or more, which determines the dimension of a bipolar vector for the character pattern.

We will use, for simplicity, a 3x3 grid for character patterns in our example. This means the Hopfield network has 9 neurons in the only layer in the network. Again for simplicity, we use two exemplar patterns, or reference patterns, which are given in Figure 2.13. Consider the pattern on the left as a representation of the character "plus", +, and the one on the right that of "minus", - .

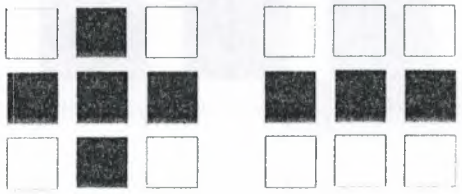

Figure 2.13. The "plus" pattern and "minus" pattern.

The bipolar vectors that represent the characters in the figure, reading the character pixel patterns row by row, left to right, and top to bottom, with a 1 for black and -1 for white pixels, are C+ = (-1, 1, -1, 1, 1, 1, -1, 1, -1), and C- = (-1, -1, -1, 1, 1, 1, -1, -1, -1). The weight matrix W is:

$$W = \begin{bmatrix} 0 & 0 & 2 & -2 & -2 & -2 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & -2 & -2 & 2 & 0 & 2 \\ 2 & 0 & -2 & 0 & 2 & 2 & -2 & 0 & -2 \\ 2 & 0 & -2 & 2 & 0 & 2 & -2 & 0 & -2 \\ 2 & 0 & -2 & 2 & 2 & 0 & -2 & 0 & -2 \\ 2 & 0 & 2 & -2 & -2 & -2 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & -2 & -2 & -2 & 2 & 0 & 0 \end{bmatrix}$$

The activations with input C+ are given by the vector (-12, 2, -12, 12, 12, 12, -12, 2, -12). With input C-, the activations vector is (-12, -2, -12, 12, 12, 12, -12, -2, -12).

When this Hopfield network uses the threshold function

$$f(x) = \begin{cases} 1 & \text{if } x >= 0 \\ -1 & \text{if } x \text{ [le] } 0 \end{cases}$$

the corresponding outputs will be C+ and C-, respectively, showing the stable recall of the exemplar vectors, and establishing an autoassociation for them. When the output vectors are used to construct the corresponding characters, you get the original character patterns.

Let us now input the character pattern in Figure 2.14

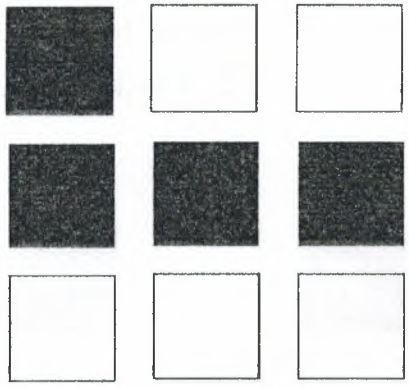

Figure 2.14. Corrupted "minus" pattern

We will call the corresponding bipolar vector **A** = (1, -1, -1, 1, 1, 1, -1, -1, -1). You get the activation vector (-12, -2, -8, 4, 4, 4, -8, -2, -8) giving the output vector, **C**- = (-1, -1, -1, 1, 1, 1, -1, -1, -1). In other words, the character -, corrupted slightly, is recalled as the character - by the Hopfield network. The intended pattern is recognized.

We now input a bipolar vector that is different from the vectors corresponding to the exemplars, and see whether the network can store the corresponding pattern. The vector we choose is **B** = (1, -1, 1, -1, -1, -1, 1, -1, 1). The corresponding neuron activations are given by

the vector (12, -2, 12, -4, -4, -4, 12, -2, 12) which causes the output to be the vector (1, -1, 1, -1, -1, -1, 1, -1, 1), same as B. An additional pattern, which is a 3x3 grid with only the corner pixels black, as shown in Figure 2.15, is also recalled since it is autoassociated, by this Hopfield network

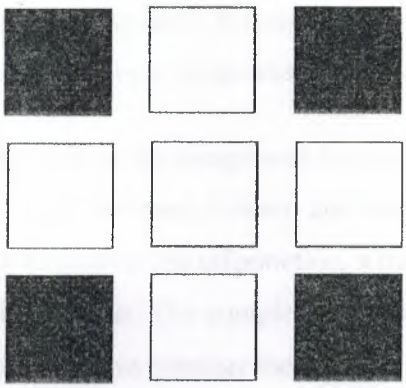

Figure 2.15. Pattern result

If we omit part of the pattern in Figure 2.15, leaving only the top corners black, as in Figure 2.16, we get the bipolar vector **D** = (1, -1, 1, -1, -1, -1, -1, -1, -1). You can consider this also as an incomplete or corrupted version of the pattern in Figure 2.15. The network activations turn out to be (4, -2, 4, -4, -4, -4, 8, -2, 8) and give the output (1, -1, 1, -1, -1, -1, 1, -1, 1), which is B.

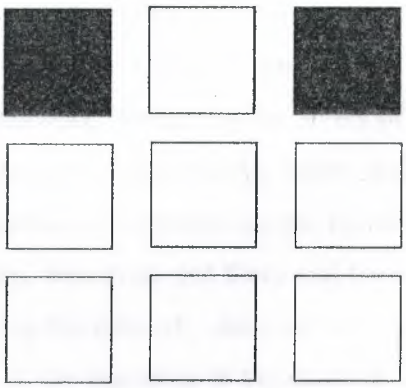

Figure 2.16. A partly lost Pattern of Figure 2.15.

2.14 Advantage and Disadvantage of Neural Networks

Artificial Neural Networks has several advantages and disadvantages. Because A.N.N. is similar to B.N.N, if parts of the network are damaged, it can still carry on its works. Another advantage is it ability to learn from limited sets of examples. However, unlike traditional program, it parts of the program are damaged, it could no longer function. Furthermore, the same neural network can be used for several programs without any modification.

The speed of the A.N.N. can be both its advantage and disadvantage. Depending on the level of AI required, a network with a larger input, hidden, and output layers may be required. If the computer is not fast enough to process the information, a tremendous amount of time may be required to process a simple question. The complexity of the network is considered to be its disadvantage because you do not know whether the network has "cheated" or not. Because a neural network can memorize and recognize patterns, it is almost impossible to find out how the network comes up with its answers. This is also known as a black box model. For example, you can provide a neural network with several pictures of a person and ask it to recognize him/her. Due to the problem just described, it is essential test network after its training by introducing it to other inputs that network has never experienced.

2.15 Summary

In this chapter we introduced a neural network as a collection of processing elements distributed over a finite number of layers and interconnected with positive or negative weights, depending on whether cooperation or competition (or inhibition) is intended. The activation of a neuron is basically a weighted sum of its inputs. A threshold function determines the output of the network. There may be layers of neurons in between the input layer and the output layer, and some such middle layers are referred to as hidden layers, others by names such as Grossberg or Kohonen layers, named after the researchers Stephen Grossberg and Teuvo Kohonen, who proposed them and their function. Modification of the weights is the process of training the network, and a network subject to this process is said to be learning during that phase of the operation of the network. In some network operations, a feedback operation is used in which the current output is treated as modified input to the same network.

A couple of examples of a Hopfield network, one of them for pattern recognition.

Neural networks can be used for problems that can't be solved with a known formula and for problems with incomplete or noisy data. Neural networks seem to have the capacity to recognize patterns in the data presented to it, and are thus useful in many types of pattern recognition problems.

CHAPTER THREE

PATTERN RECOGNITION WITH SUPERVISED LEARNING

3.1 Overview

In this chapter, I will describe my application of Pattern recognition using Neural Network Back propagation technique. This process starts with training, defining an error measure, and the flow chart of the back propagation. Then describe the source code of the application. Two data files test.dat which have the test patterns and learn.dat which have the target patterns also given in this chapter. At the end, the application layout and the procedure that how to use this application is also explained.

3.2 Training with Back Propagation

To begin discussion of training the network we must first recognize the need for a measure of how close the network has come to an established desired value. This measure is the network error. Since we are dealing with supervised training, the desired value is known to us for the given training set. The proper selection of a training set will be a crucial factor in any successful network application. The training set must be of an appropriate size and it must be reasonably well representative of the problem space. For now we assume that such a training set exists so we may interrogate how to use it to train a network.

3.2.1 Defining an error measure

Typically for the back propagation (Rumelhart et al., 1986)[18] training algorithm, an error measure known as the mean square error is used. This is in fact not a requirement. Any continuously differentiable error function can be used, but the choice of another error function does add additional complexity and should be approached with a certain amount of caution. Whatever function is chosen for the error function must provide a meaningful measure of the "distance" between desired and actual outputs of the network. The mean square error is defined as follows:

$$E_p = \frac{1}{2}\sum_{j=1}^{N}\left(t_{pj} - o_{pj}\right)^2 \qquad (3.1)$$

Where E_p is the error for the p^{th} presentation vector; t_{pj} is the desired value for the j^{th} output neuron (i.e., the training set value); and o_{pj} is the actual output of the j^{th} output neuron.

Each term in the sum is the error contribution of a single output neuron. By taking the square of the absolute error, the difference between desired and actual, we cause outputs that are distant from the desired value to contribute most strongly to the total error. Increasing the exponent, if we chose to do what, would augment this effect.

Back propagation is one of the simpler members of a family of training algorithms collectively termed gradient descent. The idea is to minimize the network total error by adjusting the weights. Gradient descent, sometimes known as the method of steepest descent, provides a means of doing this. Each weight may be thought of as a dimension in an N-dimensional error space. In error space the weights act as independent variables and the shape of the corresponding error surface is determined by the error function in combination with the training set.

The negative gradient of the error function with respect to the weights then points in the direction which will most quickly reduce the error function. If we move along this vector in weight space, we will ultimately reach a minimum at which the gradient becomes zero. Unfortunately this may be a local minimum, Figure 3.1 illustrates the operation of the gradient in the context of a two-dimensional cross section of the error space.

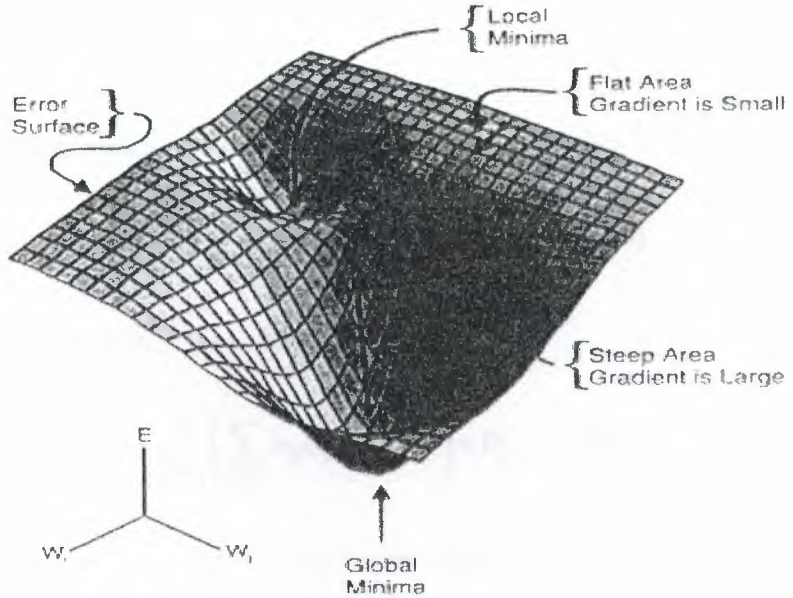

Figure 3.1. Surface gradient diagram

We can express the above observations mathematically as:

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}} \qquad (3.2)$$

The term, $\Delta_p W_{ji}$, designates the change in the weight connecting a *source* neuron, *i*, in layer L-1 and a *destination* neuron, *j*, in layer L. This change in the weight results in a step in the weight space (Figure 3.1) toward lower error.

The objective is to determine how we must adjust each weight to achieve convergence for the network. Equation 3.2 states that the change in each weight w_{ij} will be along the negative gradient leading to a steepest descent along the local error surface.

The task now is to convert equation 3.2 into a difference equation suitable for use in a computer implementation. To accomplish this evaluate the partial derivative, $\partial E_p/\partial w_{ij}$. begin by applying the chain rule:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial \text{net}_{pj}} \frac{\partial \text{net}_{pj}}{\partial w_{ji}} \qquad (3.3)$$

However, we know that net_{pj} is given by:

$$\text{net}_{pj} = \sum_i w_{ji} O_{pi} \qquad (3.4)$$

Where the sum in equation 3.4 is taken over the output, O_{pi}, of all neurons in the L-1 layer. We may therefore evaluate $\partial \text{net}_{pj}/\partial w_{ji}$, the second term in equation 3.3, as follows

$$\frac{\partial \text{net}_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} O_{pi} \qquad (3.5)$$

By expanding equation 3.5 we obtain:

$$\frac{\partial \text{net}_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{i' \neq i} w_{ji'} O_{pi'} + w_{ji} O_{pi} \right) = O_{pi} \qquad (3.6)$$

Substituting equation 3.6 into equation 3.3 we obtain:

$$\frac{\partial E_p}{\partial w_{ji}} = O_{pi} \frac{\partial E_p}{\partial \text{net}_{pj}} \qquad (3.7)$$

Now we define the error signal δ_{pj} as:

$$\delta_{pj} = -\frac{\partial E_p}{\partial \text{net}_{pj}} \qquad \textbf{(3.8)}$$

By combining equations 3.7 and 3.8 we have:

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} O_{pi} \qquad \textbf{(3.9)}$$

We may rewrite equation 3.2 by substituting equation 3.9 and supplying a constant of proportionality, η.

$$\Delta_p w_{ji} = \eta \delta_{pj} O_{pi} \qquad \textbf{(3.10)}$$

The constant η is known as the learning rate. As its name implies, it governs the distance traveled in the direction of the negative gradient when a step in weight space is taken.

In order to achieve a usable difference equation, the task of evaluation δ_{pj} still remains. Once again we must apply the chain rule:

$$\delta_{pj} = -\frac{\partial E_p}{\partial \text{net}_{pj}} = -\frac{\partial E_p}{\partial O_{pj}} \frac{\partial O_{pj}}{\partial \text{net}_{pj}} \qquad \textbf{(3.11)}$$

Now recall that the output O_{pj} is directly a function of net_{pj} as follows:

$$O_{pj} = f\left(\text{net}_{pj}\right) \qquad \textbf{(3.12)}$$

$$\frac{\partial O_{pj}}{\partial \text{net}_{pj}} = f'\left(\text{net}_{pj}\right) \qquad \textbf{(3.13)}$$

where $f()$ is the squashing function.

To evaluate $\partial E_p / \partial O_{pj}$ (the first term of equation 3.11), we must consider two cases individually:

1. The destination neuron j is an output neuron.
2. The destination neuron j is a hidden layer neuron.

For a destination neuron j in the output layer we have direct access to the error E_p as a function of O_{pj}. Therefore we write:

$$\frac{\partial E_p}{\partial O_{pj}} = \frac{\partial}{\partial O_{pj}} \left(\frac{1}{2} \sum_{j'} \left(t_{pj'} - O_{pj'} \right)^2 \right) = -\left(t_{pj} - O_{pj} \right) \qquad \textbf{(3.14)}$$

With equation 3.14 we have specialized the algorithm to the specific error function. An alternate choice of error function will lead to a different difference equation. Substituting equations 3.13 and 3.14 into equation 3.11 we may now write δ_{pj} (for destination neurons in the output layer) as:

$$\delta_{pj} = \left(t_{pj} - O_{pj}\right) f'\left(\text{net}_{pj}\right) \qquad \textbf{(3.15)}$$

For destination neurons that reside in hidden layers we cannot differentiate the error function directly. Therefore we must once again apply the chain rule to obtain

$$\frac{\partial E_p}{\partial O_{pj}} = \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} \frac{\partial \text{net}_{pk}}{\partial O_{pj}} \qquad \textbf{(3.16)}$$

In equation 3.16 the sum k is over all neurons in the L + 1 layer. Recalling the definition of net_{pk}, we may evaluate the second factor in equation 3.16 as follows:

$$\begin{aligned} \frac{\partial \text{net}_{pk}}{\partial O_{pj}} &= \frac{\partial}{\partial O_{pj}} \left(\sum_l w_{kl} O_{pl} \right) \\ &= \frac{\partial}{\partial O_{pj}} \left(\sum_{l' \neq j} w_{kl'} O_{pl'} + w_{kj} O_{pj} \right) \\ &= w_{kj} \end{aligned} \qquad \textbf{(3.17)}$$

Substituting equation 3.17 back into equation 3.16 yields:

$$\frac{\partial E_p}{\partial O_{pj}} = \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} w_{kj} \qquad \textbf{(3.18)}$$

Now we have it by definition that:

$$\delta_{pk} = p \frac{\partial E_p}{\partial \text{net}_{pk}} \qquad \textbf{(3.19)}$$

Substituting equation 3.19 into equation 3.18 yields:

$$\frac{\partial E_p}{\partial O_{pj}} = \sum_k \delta_{pk} w_{kj} \qquad \textbf{(3.20)}$$

Finally combining equation 3.11, 3.13, and 3.20 we can represent the error signal d_{pj} for hidden layers as:

$$\delta_{pj} = f'(\text{net}_{pj}) \sum_{k} \delta_{pk} w_{kj} \qquad \textbf{(3.21)}$$

To summarize the results so far, equation 3.10 provides the difference equation in terms of δ_{pj}. This is valid for both hidden and output layer weights. Equations 3.15 and 3.21 specify δ_{pj} for the output layer and hidden layer weights, respectively. Equation 3.14 particularized our solution to the mean square error. Therefore to use an alternative error function equation 3.14 would require modification. To obtain a difference equation suitable for use on a digital computer it now only remains to evaluate $f(\text{net}_{pj})$. To do this we must again particularize our solution by choosing a specific squashing function $f'(\text{net}_{pj})$. We now proceed using the sigmoid function as follows:

$$O_{pj} = f(\text{net}_{pj}) = \frac{1}{1+e^{-\text{net}_{pj}+\theta}} \qquad \textbf{(3.22)}$$

From equations 3.13 and 3.22 we may write $f'(\text{net}_{pj})$ as:

$$f'(\text{net}_{pj}) = \frac{\partial}{\partial \text{net}_{pj}} \left(\frac{1}{1+e^{-\text{net}_{pj}+\theta}} \right) \qquad \textbf{(3.23)}$$

Evaluating the derivative in equation 3.23 leads to:

$$f'(\text{net}_{pj}) = \left(\frac{-1}{\left(1+e^{-\text{net}_{pj}+\theta}\right)^2} \right) \frac{\partial}{\partial \text{net}_{pj}} \left(1+e^{-\text{net}_{pj}+\theta}\right) \qquad \textbf{(3.24)}$$

Continue the evaluation of $f'(\text{net}_{pj})$ as follows:

$$f'(\text{net}_{pj}) = \left(\frac{-1}{\left(1+e^{-\text{net}_{pj}+\theta}\right)^2}\right) e^{-\text{net}_{pj}+\theta} \frac{\partial}{\partial \text{net}_{pj}}\left(-\text{net}_{pj}+\theta\right) \quad (3.25)$$

$$= \left(\frac{1}{1+e^{-\text{net}_{pj}+\theta}}\right)\left(\frac{e^{-\text{net}_{pj}+\theta}}{1+e^{-\text{net}_{pj}+\theta}}\right) \quad (3.26)$$

$$= \left(\frac{1}{1+e^{-\text{net}_{pj}+\theta}}\right)\left(\frac{1+e^{-\text{net}_{pj}+\theta}}{1+e^{-\text{net}_{pj}+\theta}} - \frac{1}{1+e^{-\text{net}_{pj}+\theta}}\right) \quad (3.27)$$

$$= \left(\frac{1}{1+e^{-\text{net}_{pj}+\theta}}\right)\left(1-\frac{1}{1+e^{-\text{net}_{pj}+\theta}}\right) \quad (3.28)$$

We may now express f' (net_{pj}) in term of O_{pj} by substituting equation 3.22 into equation 3.28. We then obtain:

$$f'(\text{net}_{pj}) = O_{pj}(1-O_{pj}) \quad (3.29)$$

Taken together equations 3.10, 3.15, 3.21, and 3.29 provide all that is necessary to write the difference equation needed to implement training by back propagation on a digital computer where the error function is the mean square error and the squashing function is the sigmoid. As we have proceeded through this derivation we have taken pains to show the points at which modifications would be required for alternative error or activation functions.

To summarize, the difference equation required for back-propagation training is

$$\Delta w_{ji} = \eta \delta_{pj} O_{pi} \quad (3.30)$$

Where η refers to the learning rate; δ_{pj} refers to the error signal at neuron j in layer L; and O_{pi} refers to the output of neuron i in layer L-1.

With the error signal, δ_{pj}, given by:

$$\delta_{pj} = (t_{pj} - O_{pj}) O_{pj}(1-O_{pj}) \quad \text{for output neurons} \quad (3.31)$$

$$\delta_{pj} = O_{pj}(1-O_{pj}) \sum_k \delta_{pk} w_{kj} \quad \text{for hidden neurons} \quad (3.32)$$

Where O_{pj} refers to layer L; O_{pi} refers to layer L - 1; and δ_{pk} refers to layer L + 1.

True gradient descent would proceed in infinitesimal steps along the direction established by the gradient. Since this is obviously impractical for our purposes, the learning rate, η, is defined (equation 3.30). It can be seen that equation 3.30 results in a finite step size in the direction of the gradient. Here η is a constant which acts like a gain to determine the step size. The idea is to choose η large enough to cause the network to converge quickly without introducing overshoot and therefore oscillations. Later we will look at the conjugate gradient technique which in effect uses the rate at which the gradient is changing to establish a step size. Understanding the effect of the learning rate can help to choose its value judiciously. Even so, a certain amount of experimental tuning is generally required to optimize this parameter.

For clarity, the application of equations 3.30, 3.31, and 3.32 is shown in Figure 3.2. In particular, this figure is useful to clarify which layers are involved when calculating the various components of the difference equation. The top half of the figure delineates the training of the output layers. The bottom half depicts training in hidden layers. A cautionary reminder is that the above difference equation is valid only for mean square error with a sigmoidal activation function. To use alternate error or activation functions with back propagation the difference equation must be modified as shown in the derivation.

In practice a momentum term is frequently added to equation 3.30 as an aid to more rapid convergence in certain problem domains. The momentum takes into account the effect of past weight changes. The momentum constant, α, determines the emphasis to place on this term. Momentum has the effect of smoothing the error-surface in weight space by filtering out high-frequency variations. The weights are adjusted in the presence of momentum by:

$$\Delta w_{ji}(n+1) = \eta\left(\delta_{pj} O_{pi}\right) + \alpha \Delta w_{ji}(n) \qquad \textbf{(3.33)}$$

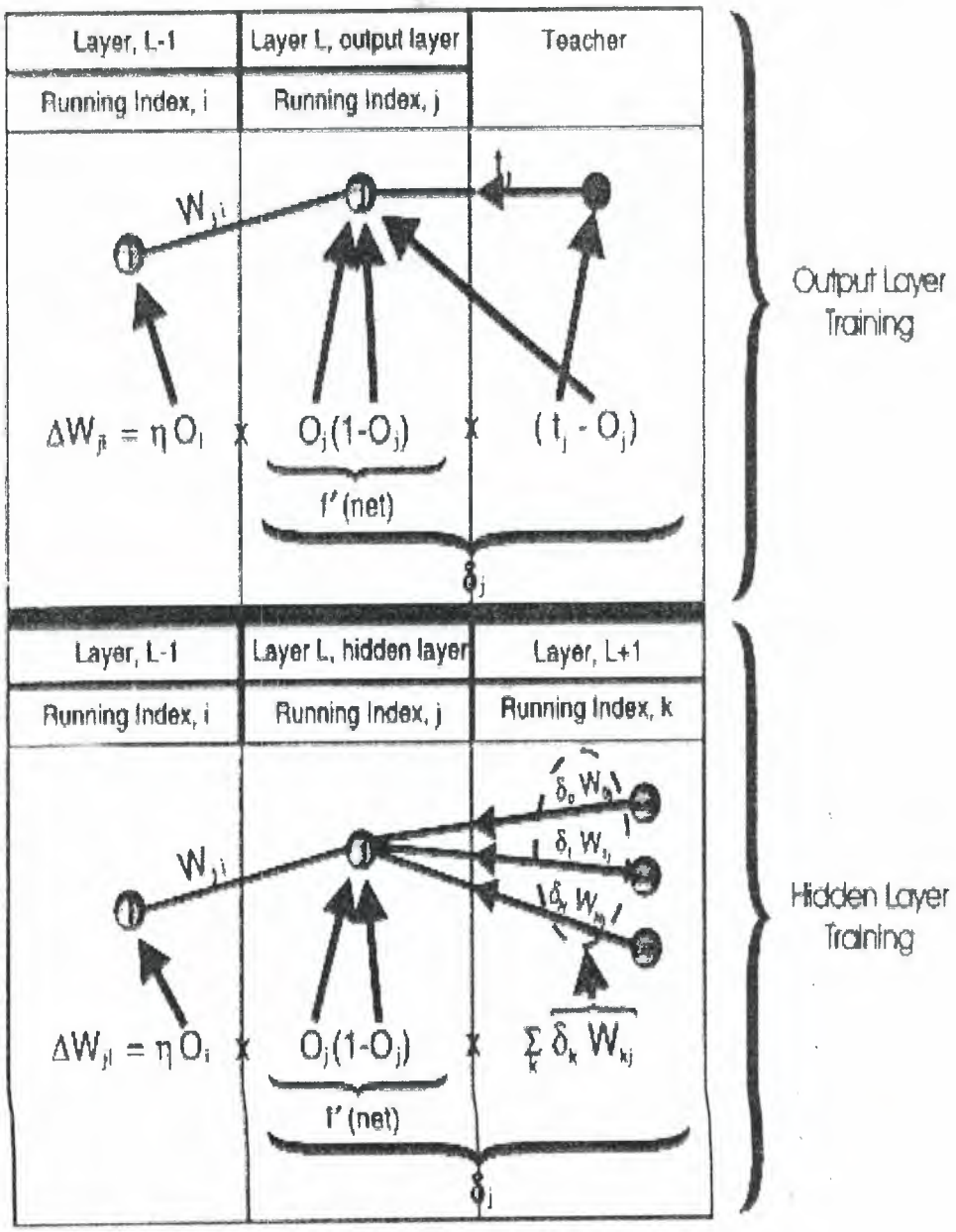

Figure 3.2. Back-propagation interaction between layers

The momentum term is but the first of several departures from what might be described as pure gradient descent that is intended to augment the algorithm with respect to its ability to converge more rapidly. At this point we remark that there is much in back propagation and its variations that are largely empirical.

3.3 Back-Propagation flow chart

The overall process of back-propagation learning including both the forward and backward pass is presented in Figure 3.2 above. To apply the back-propagation algorithm the network weights must first be initialized to small random values. It is important to make the initial weights "small". Choosing initial weights too large will make the network unattainable. After initialization, training set vectors are then applied to the network. Running the network forward will yield a set of actual values. Back propagation can then be utilized to establish a new set of weights. The total error should decrease over the course of much such iteration. If it does not, an adjustment to the training parameters, α and η, may be required. (Contradictory data, a training vector duplicated with an oppositely sensed desired value, will inhibit the network ability to converge irrespective of training parameter. In the event that extreme difficulty is encountered, verification of the training data set can prove to be worthwhile.)

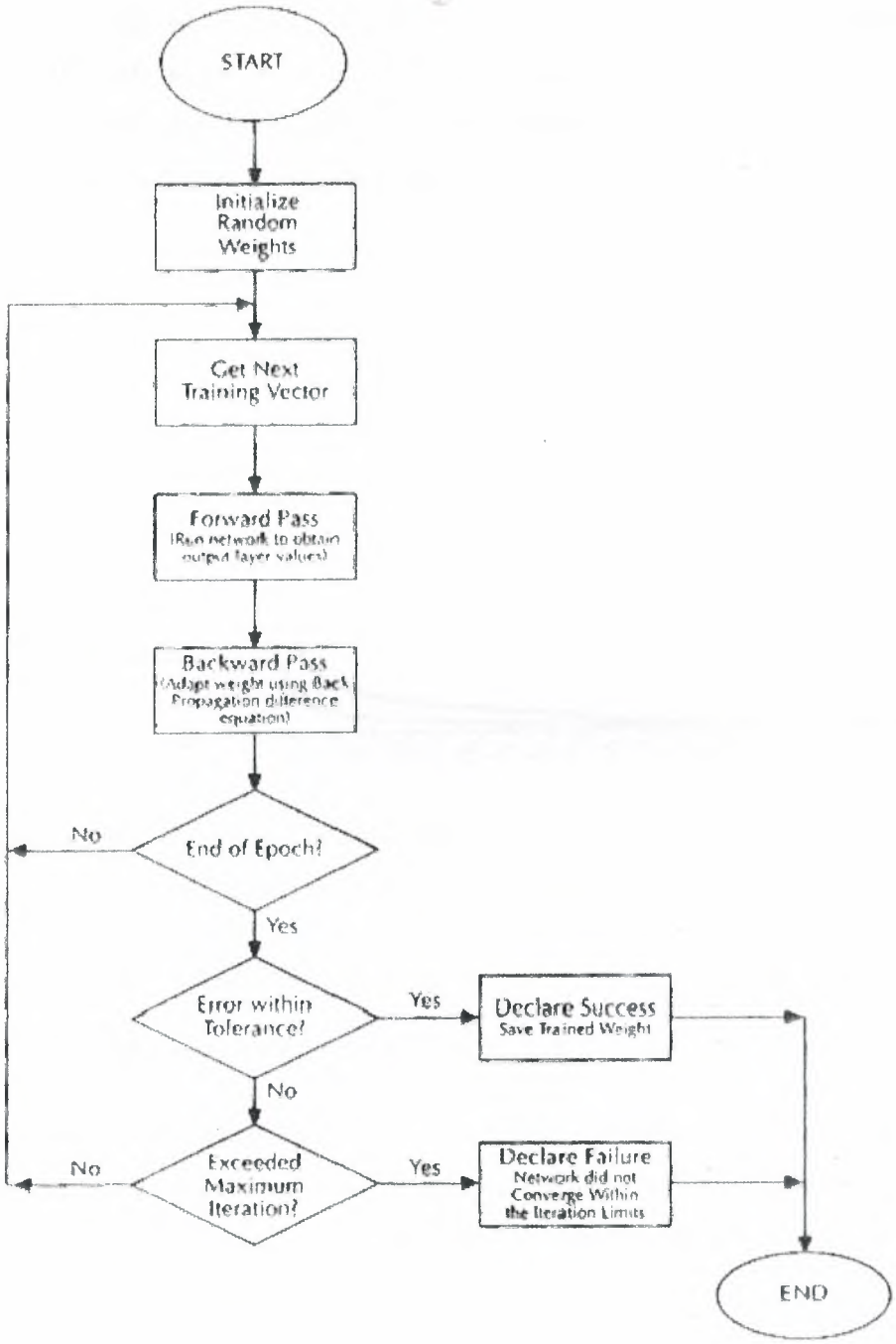

Figure 3.3. Back-propagation flow chart

One full presentation of all the vectors in the training set is termed an epoch. When the weights approach values such that the total network error, over a full epoch, falls below a preestablished threshold, the network is said to have converged. The error does not fall

necessarily uniformly. Local fluctuations in the total network error are normal and expected, especially early in the training cycle. It is useful to look at error profiles as a function of iteration to gain insight into the convergence. Figure 3.4 shown below illustrates convergence behavior during a typical training cycle.

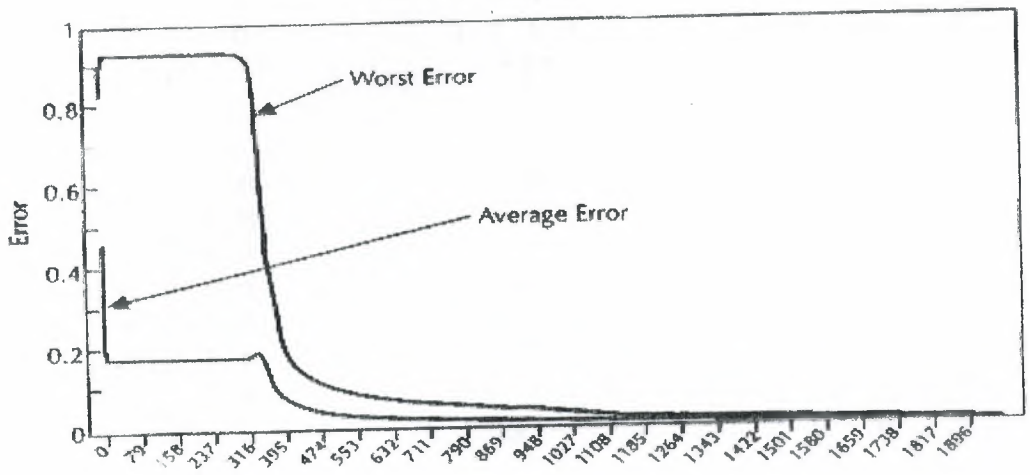

Figure 3.4. Back-propagation convergence curves. The worst error is the maximum error over a full epoch. The average error is the average over a full epoch.

3.4 Code implementing using C++

Now we will look at the C++ code implementing back propagation. The representation of the network in this code is much more basic. This simplicity will allow us to focus on the subject at hand, training with back propagation. The program source code listing is shown in APPENDIX A on page 72.

3.5 Data Files

In our program there are two dat files one is know as target file (Learn.dat), this file is consists of different patterns that are 1-9 digits and a letter "A". All patterns are made using the 5 X 7 matrix. These patterns are learnt to the program and after learning we want to check this network. For this purpose we made a file(Test.dat) this file is also consists of patterns 1-9 and a letter "A" but these all patterns are uncompleted patterns we can say them noisy patterns. Now we check our network efficiency using this test file.

- **File Learn.dat**

This file have the target patterns of 1-9 and letter "A"

```
0 0 1 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0
```

"1"

```
0 1 1 1 0
1 0 0 0 1
0 0 0 0 1
0 0 1 1 0
0 1 1 0 0
1 0 0 0 0
1 1 1 1 1
```

"2"

```
0 1 1 1 0
1 0 0 0 1
0 0 0 0 1
0 0 1 1 0
0 0 0 0 1
1 0 0 0 1
0 1 1 1 0
```

"3"

```
0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 1 1 0
0 0 0 1 0
0 0 0 1 0
0 0 0 1 0
```

"4"

```
1 1 1 1 1
1 0 0 0 0
1 1 1 1 0
0 0 0 0 1
0 0 0 0 1
1 0 0 0 1
0 1 1 1 0
```

"5"

```
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
0 1 1 1 0
0 1 0 0 1
0 1 0 0 1
0 0 1 1 0
```

"6"

```
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
```

"7"

```
0 0 1 1 0
0 1 0 0 1
0 1 0 0 1
0 0 1 1 0
0 1 0 0 1
0 1 0 0 1
0 0 1 1 0
```

"8"

```
0 0 1 1 0
0 1 0 0 1
0 1 0 0 1
0 0 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
```

"9"

```
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
```

"A"

- **File Test.dat**

This file has test patterns which are noisy patterns to check the trained Network.

0 0 1 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0
0 0 1 0 0

"Noisy 1"

0 0 0 0 0
1 0 0 0 1
0 0 0 0 1
0 0 1 1 0
0 1 1 0 0
1 0 0 0 0
1 1 1 1 1

"Noisy 2"

1 0 1 1 0
0 0 0 0 1
0 0 0 0 1
1 0 1 1 0
0 0 0 0 1
0 0 0 0 1
1 0 1 1 0

"Noisy 3"

0 1 0 1 0
0 1 0 1 0
0 1 0 1 0
0 1 1 1 0
0 0 0 1 0
0 0 0 1 0
0 0 0 1 0

"Noisy 4"

1 1 1 1 1
0 0 0 0 0
1 1 1 1 0
0 0 1 0 1
0 0 0 1 1
1 0 0 0 1
0 1 1 1 0

"Noisy 5"

0 0 0 1 0
0 0 0 0 0
0 0 0 0 0
0 1 1 1 0
0 1 0 0 1
0 1 0 0 1
0 0 1 1 0

"Noisy 6"

0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 1 1 1 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0

"Noisy 7"

0 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0
0 0 0 0 1
0 1 0 0 1
0 0 1 1 0

"Noisy 8"

0 0 1 1 0
0 1 1 1 1
0 1 1 1 1
0 0 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0

"Noisy 9"

0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1

"Noisy A"

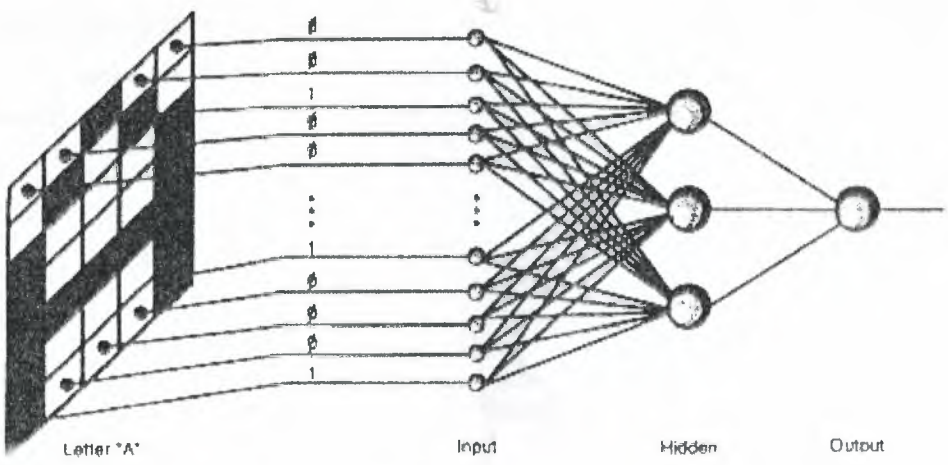

Figure 3.5. Character bit map "A" feeding into network

3.6 Software operating procedure

This software is limited to the ten different patterns 1-9 and a letter 'A'. It requires two data files, learn.dat for the numbers to learn and test.dat for testing. Both files get read on startup, and stored in arrays so you can browse them graphically without doing and training or testing.

To look at the data it will learn and test, use the spin buttons under the "Training" and *"Classify"* buttons (Figure 3.6) to browse through the testing buttons. Now, to train the perceptrons to recognize the data, merely click on *"Train"* and the learning process will start. With the data set provide with the program, the iterations should not really get much higher than 20. As a safe-guard, the program will time-out after 500 iterations, we can also change the data files.

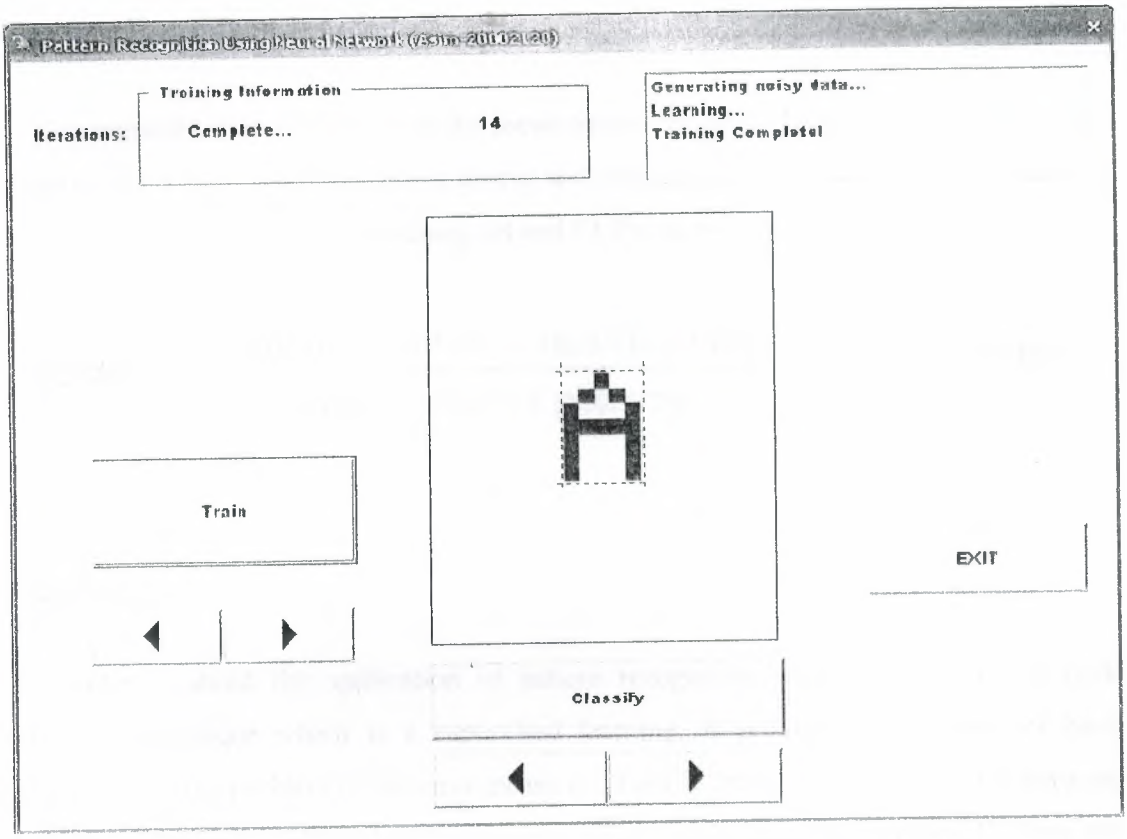

Figure 3.6. Layout of the application software

When learning has finished, the *"Classify"* button will enable. To then test what has been learnt, use the spin buttons under the *"Classify"* button to select a data entry that you would like to test. Press "Classify", and the program will write what it thinks the number is underneath. *Note:* I supplied a test set that produces very good results - with one exception. Try testing the '3' a few times, and you will find the perceptron will occasionally think it's an eight.

The program is VERY easy to expand if you want too (even without understand the code). All you will need to do is change the NN_NUMBERS define clause (in OCRDlg.h) and change the data tests. You might also want to edit the "GetText" function to return the necessary results.

3.6 Results

The convergence criteria require that the mean square for all patterns be less than the error tolerance over a full epoch. For now a simple and intuitive metric is the accuracy. I achieved recognition rates of 100% on the training set and 84.3% on the test set.

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Total number of patterns}} \qquad (3.33)$$

3.6 Summary

The chapter is about the application of pattern recognition using neural network back propagation technique which is a supervised learning. It provides the training of back propagation and the problem of the error measure solved in detail. A flowchart of the network and the implementation using the C++ programming language is also explained. There are two data files use in the program one is target file consist of 10 different patterns. These patterns are given to the network and after learning the network accuracy can be tested using the test.dat file. This test set produces very good results. Try testing the '3' a few times, and you will find the perceptron will occasionally think it's an eight. The successful results of the trained network are more than 80%.

CONCLUSION

In this project I have tried to give an overview of the Pattern Recognition System using neural networks. It's a big topic and there are many details missing but I believe, however, that I gave the general flavor of Pattern Recognition and the philosophy behind its design.

In pattern recognition, one is interested in techniques to capture the human ability to recognise and classify patterns. The basic assumption in this field is that examples can be characterised (uniquely) by a set of (relevant) measurements, called features. After measuring those particular features, a sample can be classified by inspecting the measured feature values. Pattern classification systems are two types superevised and unsupervised. There are three approaches of pattern recognition systm; Statistical, Syntactic or structural and artificial neural networks.

Introduced a neural network as a collection of processing elements distributed over a finite number of layers and interconnected with positive or negative weights, depending on whether cooperation or competition (or inhibition) is intended. The activation of a neuron is basically a weighted sum of its inputs. A threshold function determines the output of the network. There may be layers of neurons in between the input layer and the output layer, and some such middle layers are referred to as hidden layers, others by names such as Grossberg or Kohonen layers, named after the researchers Stephen.

Back Propagation technique which is a supervised learning is the one of the best technique to train the network. In my application I used this technique and trained my network with a data file *learn.dat*. That file has ten different patterns after training I check the trained network with *test.dat*. I got successful results, the recognition rates of 100% on the training set and 84.3% on the test set. This concludes that it is so effective and accurate. It has many advantages and benefits, It solved lot of things seem to be easy.

REFERENCES

[1] Chen, C. H., *Statistical Pattern Recognition*, Hayden, Washington, D.C., 1973.

[2] Jain, A. K. and Chandrasekaran, B., "Dimensionality and sample size considerations in pattern recognition practice," In *Handbook of Statistics* 2, P. R. Krishnaiah and Kanal, L. N. (eds.), North-Holland, Amsterdam, 1982.

[3] Fukunaga, K. and Hayes, R. R., "Effects of sample size in classifier design," *IEEE Trans. PAMI*, PAMI-11, pp. 873-885, 1989

[4] Foley, D. H., Considerations of sample and feature size, *IEEE Trans. Inf. Theory*, IT-18, pp. 618-626, 1992.

[5] Jain, A. and Mao, J., Neural Networks and Pattern Recognition, In *Computational Intelligence: Imitating Life*, J. Zurada, R. J. Marks II, and C. J. Robinson (Eds.), IEEE Press, Piscataway, NJ, 1994.

[6] Pao, Y., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.

[7] Tou, J. T. and Gonzalez, R. C., *Pattern Recognition Principles*, Addison-Wesley, Reading, MA, 1974.

[8] Chomsky, N., "Three models for the description of language," *Proc. Group. Inform. Th.*, vol. 2, no. 2, pp. 113-124, 1956.

[9] Gonzalez, R. C. and Thomason, M. G., *Syntactic Methods in Pattern Recognition*, Addison-Wesley, Reading, MA, 1978.

[10] Rámon Y. Cajál, "Les preuves objectives de l'unit'e anatomique des cellules nurveuses," Trob. Lab. Inest. Biol. Univ. Madrid, vol. 29, pp. 1-37, 1934. (translation: Purkiss, M. V. and Fox, C. A., Madrid: Instituto "Ramon y Cajal", 1954).

[11] Churchland, P. S. and Sejnowski, T. J., *The Computational Brain*, MIT Press, Cambridge, MA, 1992.

[12] McCulloch, W. S. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115-133, 1943.

[13] Wiener, N., *Cybernetics: Or, Control and Communication in the Animal and the Machine*, John Wiley & Sons, New York, 1948.

[14] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. of Sci.*, vol. 79, pp. 2554-2558, 1982

[15] von Neumann, J., *The Computer and the Brain*, Yale University Press, New Haven, CT, 1958.

[16] Hebb, D. O., *The Organization of Behavior: A Neuropsychological Theory*, John Wiley & Sons, New York, 1949.

[17] Rochester, N., Holland, J. H., Haibt, L. H., and Duda, W. L., "Tests on a cell assembly theory of the action of the brain, using a large digital computer," *IRE Trans. Inf. Theory*, vol. IT-2, pp. 80-93, 1956.

[18]. Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Vol. 1: Foundations,* D. E. Rumelhart and J. L. McClelland (Eds.), MIT Press, Cambridge, MA, pp. 318–362, 1986.

APPENDIX A

// OnrDlg.cpp : implementation file

```
#include "stdafx.h"
#include "Onr.h"
#include "OnrDlg.h"

#include <fstream.h>
#include <time.h>
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
#include "Hyperlink.h"
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        CHyperLink m_hLink;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	DDX_Control(pDX, IDC_LINK, m_hLink);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CAboutDlg::OnInitDialog()
{	CDialog::OnInitDialog();
	m_hLink.SetURL("c:/my application");
	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// COnrDlg dialog

COnrDlg::COnrDlg(CWnd* pParent) : CDialog(COnrDlg::IDD, pParent) {
	//{{AFX_DATA_INIT(COnrDlg)
	//}}AFX_DATA_INIT
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void COnrDlg::DoDataExchange(CDataExchange* pDX) {
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(COnrDlg)
	DDX_Control(pDX, IDC_TRAINSPIN, m_Trainspin);
	DDX_Control(pDX, IDC_TESTSPIN, m_Testspin);
	DDX_Control(pDX, IDC_TRAININFO, m_TrainInfo);
	DDX_Control(pDX, IDC_INFOBOX, m_cInfoBox);
	DDX_Control(pDX, IDC_TESTWINDOW, m_TestWindow);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(COnrDlg, CDialog)
	//{{AFX_MSG_MAP(COnrDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_TRAIN, OnTrain)
	ON_BN_CLICKED(IDC_CLASSIFY, OnClassify)
	ON_NOTIFY(UDN_DELTAPOS, IDC_TRAINSPIN, OnDeltaposTrainspin)
	ON_NOTIFY(UDN_DELTAPOS, IDC_TESTSPIN, OnDeltaposTestspin)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
BOOL COnrDlg::OnInitDialog() {
        CDialog::OnInitDialog();

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL) {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty()) {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
                }
        }

        SetIcon(m_hIcon, TRUE);                         // Set big icon
        SetIcon((HICON)(LoadImage(AfxGetResourceHandle(),
                MAKEINTRESOURCE(IDR_MAINFRAME), IMAGE_ICON, 16, 16, 0)),
FALSE);


// Set the spin control ranges.

        m_Testspin.SetRange(0,NN_NUMBERS-1);
        m_Trainspin.SetRange(0,NN_NUMBERS-1);
        m_Testspin.SetPos(0);
        m_Trainspin.SetPos(0);


// Open the necessary files.
        ifstream data("learn.dat", ios::nocreate);
        ifstream test("test.dat", ios::nocreate);


// Check whether they exist.
        if (!test || !data) {
                MessageBox("No learning or test data present.", "Cannot run...",
                        MB_OK | MB_ICONERROR);
        }

// Now read them.
        for(int i=0;i<NN_NUMBERS;i++) {
                for(int j=0;j<NN_RESX * NN_RESY;j++) {
                        int onoff;
                        test >> onoff;
                        m_bTestData[i][j] = onoff;
```

```
                    data >> onoff;
                    m_bNumbers[i][j] = onoff;
                }
        }

// Set the initial pointer to the drawing data.

m_ipDrawNum = &m_bNumbers[0][0];
        return true;
}
void COnrDlg::OnSysCommand(UINT nID, LPARAM lParam) {
        if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else {
                CDialog::OnSysCommand(nID, lParam);
        }
}

void COnrDlg::OnPaint() {
        if (IsIconic())  {
                CPaintDC dc(this); // device context for painting
                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

// Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        } else {
                CPaintDC dc(this);
                CDialog::OnPaint();

                // This seems to be a rather complicated way of
                // drawing a rectangle where I want it...but it
                // works.
                CRect rect;

                m_TestWindow.GetClientRect(&rect);
                m_TestWindow.ClientToScreen(&rect);
                ScreenToClient(&rect);
                dc.FrameRect(&rect, &CBrush(RGB(0,0,0)));
                rect.DeflateRect(1,1);
                dc.FillSolidRect(rect, RGB(255,255,255));
```

```
// Now draw the number.

DrawNumber(&dc, m_ipDrawNum, &rect);
        }
}

HCURSOR COnrDlg::OnQueryDragIcon() {
        return (HCURSOR) m_hIcon;
}

//////////////////////////////////////////////////////
// "MAXIMUM SPEED" OPTIMIZATIONS CAUSE ONR TO FAIL FOR
// SOME REASON. OPTIMIZATIONS ARE RE-ENABLED AFTER NN CODE
//////////////////////////////////////////////////////

#pragma optimize( "", off )

void COnrDlg::OnTrain() {
        srand((unsigned)time(NULL));                    // Seed.
        m_cInfoBox.ResetContent();

// Initialize weights.

        memset(&m_fWeights,0,sizeof(m_fWeights));

// Now, generate some noisy data to learn from.
        m_cInfoBox.InsertString(-1, "Generating noisy data...");
        int num = 0;
        for (int i=0;i<NN_NUMBERS * NN_NOISY;i++) {
                for(int j=0;j<NN_RESX * NN_RESY;j++) {
                        if (rand() % 100 < 7) {
                                m_iNoisy[i][j] = !m_bNumbers[num][j];
                        } else m_iNoisy[i][j] = m_bNumbers[num][j];
                }
                if ((float)(i+1)/NN_NOISY == (i+1)/NN_NOISY && i != 0)
                        num++;
        }

        m_cInfoBox.InsertString(-1, "Learning...");

        RunNet(true);

        m_TrainInfo.SetWindowText("Complete...");
        m_cInfoBox.InsertString(-1,"Training Complete!");
        GetDlgItem(IDC_CLASSIFY)->EnableWindow(true);
}

void COnrDlg::DrawNumber(CDC *dc, int *cell, CRect *rect) {
```

```
// Calculate areas to centre the images, I use the
// pointer to the CRect to get the offset, since we
// are drawing directly onto the dialog box.
CPoint tl;
tl.x = rect->Width()/2 - (NN_RESX*10)/2;
tl.y = rect->Height()/2 - (NN_RESY*10)/2;

// Offset all the coordinates calculated to draw
// in our rectangle. Also remember the initial x
// coordinate.
tl.Offset(rect->TopLeft());
int ix = tl.x;

// Create those little dashed red lines. More
// cosmetic than any else...
CPen pen;
if (pen.CreatePen(PS_DOT, 1, RGB(127,0,0))) {
        CPen *pOldPen = dc->SelectObject(&pen);

        CPoint pt1 = rect->TopLeft();
        CPoint pt2 = rect->BottomRight();
        dc->MoveTo(tl.x-5, tl.y-2);
        dc->LineTo(tl.x+NN_RESX*10+5, tl.y-2);
        dc->MoveTo(tl.x-5, tl.y+10*NN_RESY+1);
        dc->LineTo(tl.x+NN_RESX*10+5, tl.y+10*NN_RESY+1);
        dc->MoveTo(tl.x-2, tl.y-7);
        dc->LineTo(tl.x-2, tl.y+NN_RESY*10+7);
        dc->MoveTo(tl.x+NN_RESX*10+2, tl.y-7);
        dc->LineTo(tl.x+NN_RESX*10+2, tl.y+NN_RESY*10+7);

        dc->SelectObject(pOldPen);
}
```

// Draw the text if necessary.

```
CRect format(tl.x+1, tl.y+NN_RESY*10+2,
                        tl.x+NN_RESX*10-1, tl.y+NN_RESY*10+18);
if (m_bDisplayString) {
        CFont fnt;
        if (fnt.CreatePointFont(80,"Arial")) {
                CFont* pOldFont = dc->SelectObject(&fnt);

                dc->SetTextColor(RGB(0,0,127));
                dc->DrawText(m_Str,format,DT_CENTER);
                dc->SelectObject(pOldFont);
        }
}
```

```
// Draw the number.
        for(int i=0;i<NN_RESX*NN_RESY;i++) {
                if ((*cell) == 1) {
                        dc->FillSolidRect(tl.x,tl.y,10,10,RGB(0,0,0));
                }
                tl.x += 10;
                if (tl.x == ix + 10*NN_RESX) {
                        tl.x = ix;
                        tl.y += 10;
                }
                cell++;
        }
}

void COnrDlg::OnClassify() {
        int pos = (BYTE)m_Testspin.GetPos();

        m_ipDrawNum = &m_bTestData[pos][0];
        InvalidateNumber();

        float d[NN_NUMBERS];
        for(int j=0;j<NN_NUMBERS;j++) {
                d[j] = 0;
                for(int k=0;k<NN_RESX * NN_RESY;k++) {
                        d[j] += m_fWeights[j][k]*m_bTestData[pos][k];
                }
        }

        int bestind = 0;
        for(j=1;j<NN_NUMBERS;j++) if (d[j] > d[bestind]) bestind = j;

// Calculate areas to draw the text.

        CRect size;
        m_TestWindow.GetClientRect(&size);
        int x = size.Width()/2  - (NN_RESX*10)/2;
        int y = size.Height()/2 - (NN_RESY*10)/2;
        CRect format(x,size.Height()-y+1, size.Width()-x, size.Height()-5);

        m_bDisplayString = true;
        GetText(m_Str, bestind+1);
}

void COnrDlg::RunNet(bool training) {

// Test the noisy data.

        float d[NN_NUMBERS];
        int cycles = 0;
        bool correct;
```

```
        do {
                correct = true;
                for(int i=0;i<NN_NUMBERS * NN_NOISY;i++) {
                        for(int j=0;j<NN_NUMBERS;j++) {
                                d[j] = 0;
                                for(int k=0;k<NN_RESX * NN_RESY;k++) {
                                        d[j] += m_fWeights[j][k]*m_iNoisy[i][k];
                                }
                        }

                        int bestind = 0;
                        for(j=1;j<NN_NUMBERS;j++) if (d[j] > d[bestind]) bestind = j;

                        int realval = (int)(i/NN_NOISY);
                        if (bestind == realval) continue;

                        if (training) {
                                CString result;
                                result.Format("Guessed %d instead of %d.",bestind, realval);
                                m_TrainInfo.SetWindowText(result);
                                correct = false;

                                for(j=0;j<NN_RESX * NN_RESY;j++) {
                                        m_fWeights[bestind][j] -= m_iNoisy[i][j];
                                        m_fWeights[realval][j] += m_iNoisy[i][j];
                                }
                        }
                }

                SetDlgItemInt(IDC_ITERATE, ++cycles);
        } while (!correct && cycles <= NN_MAXITER);

        if (cycles >= NN_MAXITER) {
                MessageBox("Training has timed-out.",
                        "Error in Training", MB_OK | MB_ICONINFORMATION);
                return;
        }
}

#pragma optimize( "", on )

void COnrDlg::OnDeltaposTrainspin(NMHDR* pNMHDR, LRESULT* pResult) {
        NM_UPDOWN* pNMUpDown = (NM_UPDOWN*)pNMHDR;
        int iPos = pNMUpDown->iPos;
        int iDelta = pNMUpDown->iDelta;

        m_bDisplayString = false;

        if ((iPos == 0 && iDelta == -1) || (iPos == NN_NUMBERS-1 && iDelta == 1)) {
```

```
            if (iDelta == -1) {
                m_ipDrawNum = &m_bNumbers[0][0];
                InvalidateNumber();
            } else {
                m_ipDrawNum = &m_bNumbers[NN_NUMBERS-1][0];
                InvalidateNumber();
            }
        } else {
            m_ipDrawNum = &m_bNumbers[iPos+iDelta][0];
            InvalidateNumber();
        }

        *pResult = 0;
}

void COnrDlg::OnDeltaposTestspin(NMHDR* pNMHDR, LRESULT* pResult) {
        NM_UPDOWN* pNMUpDown = (NM_UPDOWN*)pNMHDR;
        int iPos = pNMUpDown->iPos;
        int iDelta = pNMUpDown->iDelta;

        m_bDisplayString = false;

        if ((iPos == 0 && iDelta == -1) || (iPos == NN_NUMBERS-1 && iDelta == 1)) {
            if (iDelta == -1) {
                m_ipDrawNum = &m_bTestData[0][0];
                InvalidateNumber();
            } else {
                m_ipDrawNum = &m_bTestData[NN_NUMBERS-1][0];
                InvalidateNumber();
            }
        } else {
            m_ipDrawNum = &m_bTestData[iPos+iDelta][0];
            InvalidateNumber();
        }

        *pResult = 0;
}

void COnrDlg::GetText(CString &str, int num) {
        switch (num) {
        case 1:
            str = "One"; break;
        case 2:
            str = "Two"; break;
        case 3:
            str = "Three"; break;
        case 4:
            str = "Four"; break;
        case 5:
            str = "Five"; break;
```

```
        case 6:
                str = "Six"; break;
        case 7:
                str = "Seven"; break;
        case 8:
                str = "Eight"; break;
        case 9:
                str = "Nine"; break;
        case 10:
                str = "Letter 'A'"; break;

        default:
                str = "Undetermined";
        }
}

void COnrDlg::InvalidateNumber() {
        CRect rect;

        m_TestWindow.GetClientRect(&rect);
        m_TestWindow.ClientToScreen(&rect);
        ScreenToClient(&rect);

        InvalidateRect(rect, false);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// HyperLink.h : header file

# if !defined(AFX_HYPERLINK_H__D1625061_574B_11D1_ABBA_
00A0243D1382__INCLUDED_)
#define AFX_HYPERLINK_H__D1625061_574B_11D1_ABBA_
00A0243D1382__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CHyperLink window

class CHyperLink : public CStatic
{
// Construction/destruction
public:
        CHyperLink();
        virtual ~CHyperLink();
// Attributes
public:
```

```
// Operations
public:
        void SetURL(CString strURL);
        CString GetURL()                { return m_strURL;   }
        void SetColours(COLORREF crLinkColour, COLORREF crVisitedColour);
        COLORREF GetLinkColour()        { return m_crLinkColour; }
        COLORREF GetVisitedColour()     { return m_crVisitedColour; }

        void SetVisited(BOOL bVisited = TRUE);
        BOOL GetVisited()               { return m_bVisited; }
        void SetLinkCursor(HCURSOR hCursor) { m_hLinkCursor = hCursor; }
        void SetUnderline(BOOL bUnderline = TRUE);
        BOOL GetUnderline()                     { return m_bUnderline; }
        void SetAutoSize(BOOL bAutoSize = TRUE);
        BOOL GetAutoSize()              { return m_bAdjustToFit; }

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CHyperLink)
        public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
        protected:
        virtual void PreSubclassWindow();
        //}}AFX_VIRTUAL

// Implementation
        protected:
        HINSTANCE GotoURL(LPCTSTR url, int showcmd);
        void ReportError(int nError);
        LONG GetRegKey(HKEY key, LPCTSTR subkey, LPTSTR retdata);
        void PositionWindow();

// Protected attributes
      protected:
        COLORREF m_crLinkColour, m_crVisitedColour;     // Hyperlink colours
        BOOL    m_bVisited;
        BOOL  m_bUnderline;             //underline hyperlink?
        BOOL  m_bAdjustToFit;           // Adjust window size to fit text?
        CString  m_strURL;              // hyperlink URL
        CFont   m_Font;                 // Underline font if necessary
        HCURSOR  m_hLinkCursor;         // Cursor for hyperlink
        CToolTipCtrl m_ToolTip;         // The tooltip

        // Generated message map functions
protected:
        //{{AFX_MSG(CHyperLink)
        afx_msg HBRUSH CtlColor(CDC* pDC, UINT nCtlColor);
        afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
        //}}AFX_MSG
        afx_msg void OnClicked();
```

```
        DECLARE_MESSAGE_MAP()
};

#endif
// !defined(AFX_HYPERLINK_H__D1625061_574B_11D1_
ABBA_00A0243D1382__INCLUDED_)
////////////////////////////////////////////////////////////////////////////////////////////////////

// Onr.h : main header file for the ONR application


#if !defined(AFX_ONR_H__B68A5325_43B0_11D3_
                96EE_F0A5A92B9882__INCLUDED_)
#define AFX_ONR_H__B68A5325_43B0_11D3_96EE_
                F0A5A92B9882__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif                                                  // _MSC_VER >= 1000
#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"                                   // main symbols

////////////////////////////////////////////////////////////////////////////

// COnrApp:
// using Onr.cpp for the implementation of this class
//

class COnrApp : public CWinApp
{
public:
        COnrApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COnrApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(COnrApp)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//////////////////////////////////////////////////////////////////////

#endif   // !defined(AFX_ONR_H__B68A5325_43B0_11D3_
         //   96EE_F0A5A92B9882__INCLUDED_)


/* ============================================================== *\
 * File:          OnrDlg.h                                        *
 * Description:   Header file for the main dialog,                *
 *                has all the important functions.                *
\* ============================================================== */

#ifndef _ONRDLG_H_
#define _ONRDLG_H_

#if _MSC_VER >= 1000
#pragma once
#endif

#define NN_NUMBERS      10
#define NN_RESX         5
#define NN_RESY         7
#define NN_NOISY        25
#define NN_MAXITER      500

class COnrDlg : public CDialog {
public:
        COnrDlg(CWnd* pParent = NULL);

        //{{AFX_DATA(COnrDlg)
        enum { IDD = IDD_ONR_DIALOG };
        CSpinButtonCtrl      m_Trainspin;
        CSpinButtonCtrl      m_Testspin;
        CStatic         m_TrainInfo;
        CListBox        m_cInfoBox;
        CStatic         m_TestWindow;
        //}}AFX_DATA

        //{{AFX_VIRTUAL(COnrDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        //}}AFX_VIRTUAL

protected:
        int             m_bTestData[NN_NUMBERS][NN_RESX * NN_RESY];
        int     m_bNumbers[NN_NUMBERS][NN_RESX * NN_RESY];
        int     m_iNoisy[NN_NUMBERS * NN_NOISY][NN_RESX *
                 NN_RESY];
         int    *m_ipDrawNum;
        bool    m_bDisplayString;
```

```
        float    m_fWeights[NN_NUMBERS][NN_RESX * NN_RESY];

        HICON        m_hIcon;
        CString m_Str;

        void    InvalidateNumber();
        void    GetText(CString &, int);
        void    RunNet(bool training);
        void    DrawNumber(CDC *, int *, CRect *);

        // Generated message map functions

        //{{AFX_MSG(COnrDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnTrain();
        afx_msg void OnClassify();
        afx_msg void OnDeltaposTrainspin(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnDeltaposTestspin(NMHDR* pNMHDR, LRESULT* pResult);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////END OF PROGRAM//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```