AN ABSTRACT OF THE THESIS OF

Jin Jiang for the degree of Master of Science in Electrical and Computer Engineering presented on March 2, 1990.

Title: The Architecture and Design of a Neural Network Classifier.

Abstract approve Redacted for Privacy

John Murray

The objective of this thesis is to present the architecture and design of a neural network-based pattern classifier. The classifier detects textual characters which have been translated, rotated, and corrupted by noise. This form of pattern classifier differs significantly from traditional pattern classifiers. The neural network architecture used in implementing this classifier incorporates massive parallelism, distributed memory, fault tolerance, and is capable of learning. Traditional classifiers rarely incorporate all these features.

The classifier's neural network topology, interconnect structure, learning algorithms, test methodology, and test results are presented in the thesis.

THE ARCHITECTURE AND DESIGN OF A NEURAL NETWORK CLASSIFIER

By
Jin Jiang

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirement for the
degree of
Master of Science

Completed March 2, 1990
Commencement June 1990

APPROVED:

Redacted for Privacy

Associate Professor of Electrical and Computer Engineering, in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented March 2, 1990

ACKNOWLEDGEMENTS

I would like to express my thanks to a number of people for their help in the completion of this thesis. First I would like to give my thanks to my major advisor, Dr. John Murray, who gave me help and guidance during my entire program. I would also like to express my thanks to Professor James Herzog, Professor Paul Cull, for their fruitful suggestions.

Most of all, I wish to give my deeply thanks to my parents, whose constant support and encouragement make this work possible.

TABLE OF CONTENTS

TABLE OF FIGURES

THE ARCHITECTURE AND DESIGN OF A NEURAL NETWORK CLASSIFIER

1 INTRODUCTION

1.1 Overview

Artificial neural networks have been studied for many years in the hope of achieving enhanced performance in the general areas of pattern recognition, estimation, robotic control, and fault-tolerant computing. Neural networks have great potential in speech and image recognition applications, signal processing, industrial inspection, and economic modeling [1]. A number of neural networks have been developed that found applications in above areas.

Neural networks are composed of many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets. The computational elements or nodes are connected to each other via interconnections whose "strengths" or "weights" are typically adapted during use. The modification of these weights causes learning to occur in the network, and modifies the network in such a way as to accomplish the particular task at hand.

Instead of executing a sequential program to accomplish a task, as in the Von Neumann model of computing, neural networks explore many competing hypotheses simultaneously. Computations occur in a parallel distributed manner throughout the network.

There has been a recent resurgence in interest in the field of neural networks due to the development of new network topologies, algorithms, and VLSI implementation techniques. In addition, the belief that computing systems incorporating massive parallelism are essential to achieve the high performance required in speech and image recognition tasks has brought about increased research funding in this area.

1.2 Historical Perspective

Work on artificial neural networks has a long history. The development of detailed mathematical models for neural computing began more than 40 years ago with the works of McCullough and Pitts of The University of Illinois. McCullough and Pitts outlined how neurons communicate with each other eletrochemically, and derived an original mathematical model for neuron behavior under various stimuli [2].

In 1958 Cornell University psychologist Frank Rosenblatt used hundreds of these artificial "neurons" in creating a two-layer pattern learning neural network called the "Perceptron" [3]. The key to Rosenblatt's system was that it had the ability to learn. In the brain, learning occurs predominantly by the modification of the connections between neurons. If a neuron is active, e.g. producing an output, and is connected to the input of another neuron which is active as a result of the connection, then the connection between them will become stronger. If one of the neurons is active the other is not, then the connection between them will become weaker. This physiological mechanism is utilized by all animals during the process of learning, and was first presented formally by Donald Hebb. It is called Hebb's learning rule [4]. It was later verified experimentally. This rule was used as the basis for learning in the Perceptron.

Other more complex neural networks were built by Bernard Widrow, an electrical engineering professor at Stanford University. Widrow developed a machine called "Adaline"[5] that could translate speech and predict the weather with greater accuracy than the local weather forecaster.

In 1969, Marvin Minsky and Seymour Papert- Major forces in the AI field of MIT wrote a book called "Perceptrons" that attacked the perceptron design as being " too simple to be serious". The Problem with early neural network was that these neural networks were so limited to some simple logic operations, because they could only solve those problems in which each categories in the input set are linearly separatable. The idea of a multi-layer perceptron was proposed, but without a good multi-layer network learning rule, there was no possibility of achieving Rosenblatt's expectations for neural networks. As a result of this book neural network research went into its "dark ages". It was an inactive one until 1982.

It was John Hopfield , a professor at CalTech, who brought neural networks back to life again in the research community. A paper he wrote in 1982 described mathematically how neurons could act collectively to process and to store information, It presented a sophisticated, coherent theoretical picture of how a neural network could work [6]. This paper is credited with reinvigorating the neural network field .

The resurgence of more sophisticated neural networks was largely due to the availability of VLSI as an implementation medium, low-cost semiconductor memory, generally greater computer power, new network topologies and more sophisticated learning rules. The examples of these neural networks are Back-Propagation [7], Grossberg's machine [8], etc.

One of the most important application areas in neural network research is pattern recognition. Character recognition is a subset of pattern recognition. We know that neural computing systems handle a large amount of information at once and are often good at pattern recognition tasks.

One of the research areas within the neural network community is that recognition of characters under translation and rotation. Bernard Widrow and Rodney Winter of Stanford, and M. Hosokawa of Univ. of Tokushima, did research work on this subject. The neural network they trained could recognize letter patterns even if these patterns had been translated or rotated. They both used the same approach to solve this problem. They used a network

Preprocessor called an invariance network, the output of which is invariant to the translations or rotations of the input patterns[9][10].

Another character recognition area of great interest, although different from the subject of this thesis is the recognition of handwritten characters. Researchers at Nestor, Inc. have developed a neural computing system for recognition of handwritten letter patterns. It can recognize a variety of handwriting styles and is able to make better guesses. Other research work in this area have done by W.E. Weideman of Recognition equipment, Inc[11], K. Yamada of NEC. Co.[12], H.Y. Lee of Univ. of Maryland[13].

Complicated character can also be recognized using neural networks, like a Chinese character. This research has been performed successfully by Wuhan University[14].

1.3 Objectives

The objective of this thesis is to present the architecture and design of two multi-layer neural networks capable of performing English character recognition. The characters in question are subject to translation and rotation, but not magnification. The neural network is also capable of recognizing characters in the presence of random noise. The performance of the two architectures are compared and contrasted and simulation results are presented.

2 NEURAL NETWORK ARCHITECTURE

2.1 What is Neural Computing

2.1.1 Neural Networks and Human Physiology

The human brain is the most complex computing device known to man. The brain's powerful capabilities allow thought, memory, and problem-solving to occur. These capabilities have inspired scientists to attempt computer modeling or simulation of some portions of the brain's operation. One of the physiologically motivated areas of computation in which significant progress has been made recently is that of neural computing. Before we describe the building blocks and operation of neural networks, the heart of all neural computing systems, it is instructive to briefly examine the corresponding microstructual components of the brain which inspired neural computing. The most elementary functional unit of the nervous system, of which the brain is a part, is the individual neuron.

The neuron, illustrated in Figure 2.1, is a highly specialized individual cell. It's function is to act as a simple processing unit which receives inputs via its input terminals or dendrites from other neurons, and combines these signals to produce an appropriate output signal. If the combined signal is strong enough to exceed the neuron's "threshold" it activates the neuron. The neuron then "fires", producing an output signal or action potential.

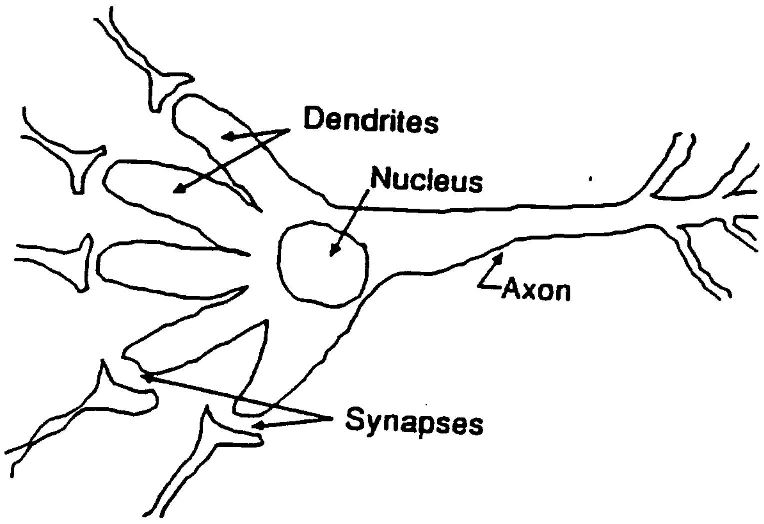


Fig. 2.1 Neuron- Building Blocks for Brain

The action potential is a pulse of approximately 1ms duration and 100mv amplitude. The output occurs on the axon of the neuron. The axon is connected to many other neurons. The connections occur as a result of the branching of the axon. If the combined signal is not strong enough to exceed the threshold level, no output pulse is generated.

The brain itself consists of approximately ten billion interconnected neurons. Each neuron is connected to one to ten thousand other neurons. The output of each neuron splits up and connects to the input of other neurons at the dendrites though a junction referred to as a synapse. The transmission of information across this junction is chemical in nature, and the magnitude of the signal transferred depends on the synaptic strength of the junction. Synaptic strength is the neural parameter which is modified in the process of learning. Synapse modification can be considered the basis for long term memory as well.

A neural network is a simplified mathematical model for the brain. It consists of many processing elements which are analogous to neurons. These processing elements are usually organized into a sequence of layers with full or random connections between successive layers. Fig 2.2 shows a simple neural network architecture. A input layer is the layer where input data is presented to the neural network. An output layer holds the response of the neural network to a given input. Layers between the input layer and output layer are called hidden layers.

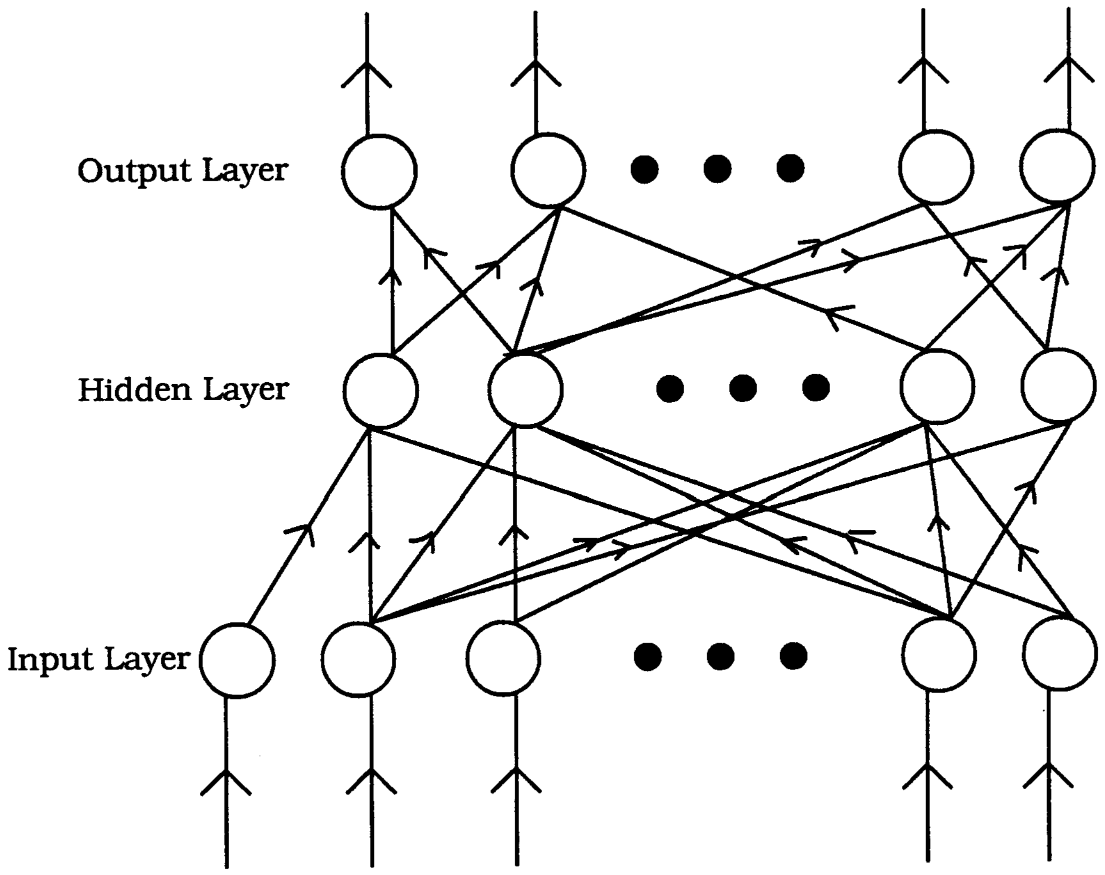


Fig 2.2 Neural Network Architecture

2.1.2 Neural Network Operation

We begin with an analysis of basic processing elements(PE's) in neural network models and then describe specific assumptions regarding these PE's.

The basic PE's of a neural network are illustrated in Figure 2.3. In the neural network, there exist a set of PE's, indicated by circles in the diagram. At each point in time, each PE u_i has an activation value, denoted in the diagram as $a_i(t)$; this activation value is passed through a function f_i to produce an output value $o_i(t)$. This output value is then passed through a set of connections to other PE's in the networks. There is associated with each connection a real number, usually called a weight or strength of the connection. This weight is designated w . The weight determines the effect that the first PE in the network has on the second. All the inputs to a given PE are multiplied by their respective weights and then combined by some operator, usually summation. The weighted summation to an PE along with its current activation, determine, via a function F, its new activation value. The whole system is viewed as being plastic in the sense that the pattern of interconnections is not fixed ; rather, the weights can undergo modification as a function of experience. The experience involves exposing the network to a set of inputs. As a result of applying theses inputs, the weights are modified according to some supervised or unsupervised learning algorithm. In this way the

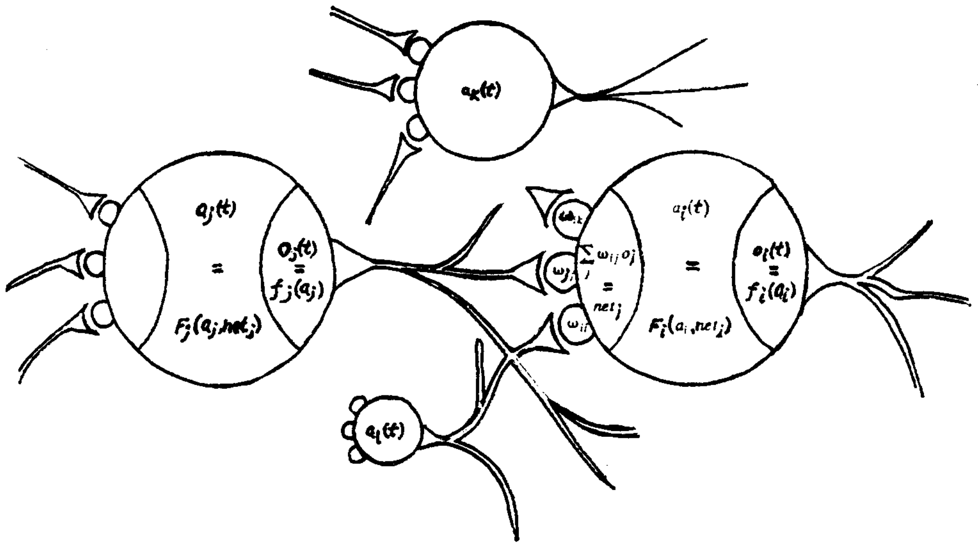

$a_k(t)$
$a_j(t) = F_j(a_j, net_j)$
$o_j(t) = f_j(a_j)$
$a_i(t) = F_i(a_i, net_i)$
$\sum_j w_{ij} o_j = net_i$
$o_i(t) = f_i(a_i)$
w_{ik}
w_{ji}
w_{il}
$a_l(t)$

Fig. 2.3 Processing Elements of Neural Network

system can evolve. We now describe some of the major elements of a neural network in detail.

2.1.2.1 A Set of Processing Elements

All neural networks include a set of PE's. Specifying the set of PE's and what external variables they process is typically the first stage in specifying a neural network. In some models these PE's may represent particular conceptual objects such as features, letters, words; in others they are simply abstract elements over which meaningful patterns can be defined. When we talk about a distributed representation [15], we mean one in which the PE's represent small, feature-like entities. In this case it is the pattern as a whole that is the meanful level of analysis. This should be contrasted to a one-unit-one-concept representation in which a single PE represents entire concepts or other larger meaningful entities.

There are many PE's in a neural network. All the processing or computing is carried out by these PE's. There is no executive processor or other manager. There are only very simple semi-autonomous processing units, each doing its own relatively simple job. An PE's job is to receive inputs from its neighbors and, as a function of the inputs, to compute an output value which it sends to its neighbors. The system is massively parallel in that many PE's can carry out their computations at the same time.

Within any system we are modeling, it is useful to characterize three types of PE's: input PE's, output PE's, and hidden PE's. Input PE's receive inputs from sources external to the neural network. Output PE's send signals out of the neural networks to other neural systems or to the motor system. Hidden PE's are those PE's whose inputs and outputs are connected internally, that is, within the neural network. They are not "visible" to the outside world.

2.1.2.2. The State of Activation.

In addition to the set of PE's, we need to create a representation of the state of the entire neural system at a given time, t. The state is specified by a vector of N real numbers $\mathbf{a}(t)$, representing the pattern of activation over the set of PE's. Each element of the vector stands for the activation of one of the PE's at time t. The activation of PE u_i at time t is designated $a_i(t)$. It is the pattern of activation over the set of PE's that captures the state of the system at any time. It is useful to observe that the processing taking place in the system may be considered as the evolution of a pattern of activity over the entire set of PE's.

Different models make different assumptions about the activation value that a PE can take on. Activation values may be continuous or discrete. If they are continuous, they may be unbounded or bounded. If they are discrete, they may take on binary values or any of a small set of values. If the PE's are continuous, then they can take on any real valued number as an activation value. In other cases, they may take on any real value between some minimum and maximum, for example, the interval [0,1]. When the activation values are restricted to discrete values, they are usually binary. Sometimes these values only take on the value 0 and 1, where 0 is taken to mean that the PE is inactive and 1 is taken to mean that it is active. As we see each of these assumptions will lead to a model with slightly different characteristics.

2.1.2.3. Processing Elements Outputs

In any neural network a number of PE's will interact. They transmit output signals to their neighbors. The strength of the signals they transmit is determined by their degree of activation. Associated with each PE, u_i , there is an output function, $f_i(a_i(t))$, which maps the current state of activation $a_i(t)$ to an output signal $o_i(t)$, that is $o_i(t) = f_i(a_i(t))$. In vector notation, we represent the current set of output values by a vector $\mathbf{o}(t)$. In some models the output value is exactly equal to the activation level of the PE. In this case f is the identity function $f(x)=x$. But for most models f is some form of threshold function, so that a one PE has no affect on another PE unless its activation value exceeds a certain threshold. Sometimes the function f is a stochastic function in which the output of the PE depends in a probabilistic fashion on its activation values.

2.1.2.4 Patterns of Connectivity and Propagation Rules.

PE's are connected to one another as described previously. It is this pattern of connectivity that constitutes exactly what the network knows and eventually determines how the network will respond to any arbitrary input. Specifying the whole neural network and the knowledge encoded therein is a matter of specifying the exact pattern of connectivity among the PE's.

In the general case, we assume that each PE provides an additive contribution to the input of the PE's to which it is connected. So the total input to a PE is simply the weighted sum of the separate inputs from each of individual PE's. That is, the inputs from all of incoming PE's are simply multiplied by a weight and then summed together to calculate the overall input to the PE. The total pattern of connectivity can be represented by specifying the weights for each of the connections in the neural system. A positive weight represents an excitatory input and a negative weight represents an inhibitory input. It is often convenient to represent such a pattern of connectivity by a weight matrix **w** in which the entry w_{ij} represents the strength and sense of the connection from PE u_i to PE u_j. The w_{ij} is a positive number if PE u_i excites PE u_j ; it is a negative number if PE u_i inhibits PE u_j; and it is 0 if there is no direct connection between PE u_i and PE u_j. The absolute value of w_{ij} specifies the strength of the connection. The relationship between the connectivity and the weight matrix is illustrated in Fig 2.4.

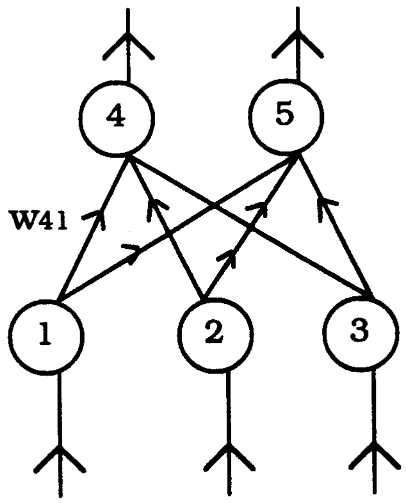


Fig 2.4a Network Structure

	u1	u2	u3	u4	u5
u1	0	0	0	W41	W51
u2	0	0	0	W42	W52
u3	0	0	0	W43	W53
u4	0	0	0	0	0
u5	0	0	0	0	0

Fig 2.4b Connection Matrix

Fig. 2.4 Neural Network Structure and Associated Connection Matrix

An additional important issue which determines both the amount information can be stored and how much serial processing the neural networks have to perform is the fan-in and fan-out of a PE. The fan-in is the number of PE's that either excite or inhibit a given PE. The fan-out of a PE is the number of PE's affected by this PE.

Now we will denote the output of a set of PE's as $\mathbf{o}(t)$. We also need a propagation rule which takes on the values of the output vector $\mathbf{o}(t)$, and combines it with the connectivity matrix to produce a net input for a specific PE. We denote the net input to PE u_i as $net_i(t)$. The propagation rule is that $net_i(t)=\mathbf{wo}(t)$. So the net input for a PE is the outputs of those who send its output to the PE, multiplied by the corresponding weights.

2.1.2.5. Activation Rule.

We also need a rule whereby the net input impinging on a particular PE is combined with the current state of the PE to produce a new state of activation. We have a function F, which takes $a_i(t)$ and the net input $net_i(t)$ and produces a new state of activation for the PE. In the simplest case, when F is the identity function, we can write $a_i(t+1)=\mathbf{wo}(t)$. If F is a threshold function, the net input must exceed some value before contributing to the new state of activation. Generally , the new state of activation depends on the old one and the current input as well. So we have $a_i(t+1)=F(a_i(t), net_i(t))$, the function F itself is what we call the activation rule.

Usually, the function is assumed to be deterministic. For example, if a threshold is involved it may be $a_i(t)=1$ if the net input exceeds the threshold value, and $a_i(t)=0$ otherwise.

2.1.2.6. Modifying the Weights as a Function of Experience.

Changing the processing or knowledge structure of the neural network system involves modifying the patterns of interconnectivity. There are mainly three kinds of modifications:

1. The development of new connections.
2. The loss of some existing connections.
3. The modification of the strengths of connections.

Case (1) and (2) can be considered as a special case of (3). Whenever we change the connection strength from zero to some positive or negative value, it has the same effect as growing a new connection. Whenever we change the connection strength to zero, it has the same effect as losing an existing connection. We will next discuss those rules whereby the strengths of these connections are modified though learning.

We will now discuss the Hebbian learning rule. Virtually all learning rules for neural models of this type can be considered a variant of the Hebbian learning rule. The following constitute the basic idea behind Hebbian learning: If an active unit, u_i, receives an input from another unit, u_j; then , if both are highly active, the weight, w_{ij}, which is the connection strength of unit i and unit j, should be strengthened. The idea can be mathematically stated as :

$$\Delta w_{ij} = g(a_i(t), t_i(t)) * h(o_j(t), w_{ij})$$

where $t_i(t)$ is a kind of teaching input to u_i. Simply stated, this equation says that the change in the connection from u_i to u_j is given by the product of a function, g(), of the activation of u_i and its teaching input t_i and another function, h(), of the output value of u_j and the connection strength w_{ij}.

Learning is the most important part of the neural network operation. we will discuss learning in detail in a later section.

2.2. A Comparison of Neural and Traditional Computing Techniques

2.2.1. Traditional Computing versus Massive Parallelism

We know that traditional computing machines execute programs in a sequential manner. It is very time consuming for some tasks to be performed using traditional computing approaches. Speech and pattern recognition tasks, for example, are required to process a large volume of information in a very short time. Using traditional Von Neumann computers,the time to process the information sequentially may be excessive. We need some computing machinery that processes large amounts of data simultaneously, then generate appropriate categorical outputs. We also require a reasonable response to noisy and incomplete input data. This is very difficult for a traditional Von Neunamm computer to do.

Neural computing systems provide us with a new way to handle such problems. Because of their architecture, they can process all the input data simultaneously (in parallel), plus they also have the ability to learn and build unique feature recognition structures for particular problems. With the computing power of neural networks, difficult computing tasks can be done a lot faster.

2.2.2. Learning by Example.

In a traditional computing system we have well defined hardware and software structures. The structure never changes or adapts during system usage. Unlike traditional Von Neumann machine where knowledge is made explicit in the form of rules or programs consisting of algorithms plus data structures, neural networks generate their own rules and their own knowledge structures by learning from the example input data presented to them. So there is always a process of adapting to variations of data presented to the network from the outside world. Learning is achieved through a learning rule which adapts or modifies the connection weights of the neural network in response to the example input and (in the case of supervised learning) the desired outputs associated with these inputs.

2.2.3 Distributed Associative Memory.

An important characteristic of neural networks is the way they store information. Unlike traditional Von Neumann computers which store data in a block of memory centrally, neural computing systems store their data and information in a distributed manner. Since the connection weights constitute the memory elements in the neural computing system. It is this kind of structure for storing data and information that makes it possible for neural computing systems to process all information simultaneously in parallel, because all the PE's store their data locally, so all PE's can get the data they need simultaneously. The value of the weights represent the current state of knowledge of the neural network. A unit of knowledge, represented for example by an input-output relationship, is distributed across all memory units in the neural network. It shares these memory units with all other units of knowledge stored in the neural network system.

Neural computing memory is associative in that if the trained neural network is presented with a partial input the neural network will choose the closest match in memory to that input, and generate an output which corresponds to a fully specified input. If the neural network is auto-associative, that is, the input is equal to the desired output for all example pairs used to train the neural network, then the presentation of partial input patterns to the neural network will result in their completion.

The distributed and associative nature of neural network memory leads to reasonable network response when it is presented with incomplete, noisy or previously unseen input pattern. The latter property is referred as generalization. The quality and meaningfulness of a generalization is dependent on the particular application and on the sophistication of the neural network. Some neural networks, like accretive neural network, map the input to the response of the closest previously learned input pattern; others like non-linear multi-layer neural networks, in particular back-propagation networks, which learn in their hidden layers important features of the knowledge domain, can use this hidden knowledge to construct non-trivial generalizations about the input data, i.e. they provide a prediction capability.

2.2.4. Fault Tolerance.

Whereas traditional Von Neumann computing systems are rendered useless by even a small amount of damage to system memory, neural computing systems are more fault tolerant. Fault tolerance refers to the fact that in most neural networks, if some elements or links are destroyed, disabled, or their connection strength changed, then the behavior of the neural network as a whole is only altered slightly. As more PE's or links are destroyed, the behavior of the neural network as a whole is degraded further. Performance will suffer in this case, but the whole computing system does not come to an abrupt halt, in general. This advantage of neural computing systems arises from three aspects of its architecture. First, because the connections of a PE to other PE's are local, damage to a single PE or some links constitutes local damages. It will not damage the whole computing system. Second, unlike traditional computing systems where all the information is stored in one place, neural networks store their information and data in a distributed way throughout the whole computing system. Global storage of information results from this distribution process. Information stored in this manner is difficult to destroy without destroying the entire system. Third, the non-linear characteristics of the PE's makes the neural networks more fault tolerant. If, for example, there are n inputs to a PE, and the weighted summation of these inputs exceeds a threshold level, causing a PE to be active. If some connections in this system are broken, it may be the case that the weighted summation still exceeds the threshold level, and the PE is still active.

The effect of some malfunctioning PE's or some broken connections will becoming less and less as the number of PE's in the system as a whole increases, either within a given layer, or across multiple layers. This characteristic of fault tolerance, or graceful degradation, makes neural computing systems extremely well suited for applications where failure of computing equipment may mean disaster: in nuclear power plant operation, missile guidance systems, space probe operations, and so on.

2.3. Adaption and Learning.

Learning in neural networks is the process of adapting the neural connection weights in response to stimuli presented at the input layer buffer. Learning can be separated into two distinct classes:

1. Associative learning, a learning technique in which the units of a network learn to produce a particular pattern of activation at the output of one set of PE's when a different pattern is presented at the input to the other set of PEs. In general, such a learning scheme must allow an arbitrary pattern at the input to one set of PE's to produce another arbitrary pattern at the output of another set of PE's. This kind of learning requires a knowledgeable teacher to compare the difference between the desired output and the actual output. The teacher must cause the connection strengths or weights to be modified if there exists a difference between the desired output values and the actual output values for a given set of inputs. This learning approach is referred to as supervised learning.

2. Regularity discovery, a learning technique in which PE's learn to response to "interesting" patterns presented at their inputs. In general, such a scheme should be able to form the basis for the development of feature detectors or structures which define input categories. Regularity discovery does not require a knowledgeable teacher, so it is referred to as unsupervised learning.

In some neural networks, these two modes of learning are mixed. This type of learning is called reinforcement learning. An external teacher indicates whether the response to an input is good or bad. It is valuable to see the different goals of the two kinds of learning. Associative learning is generally employed whenever we are concerned with storing patterns so that they can be re-evoked in the future. Associative learning rules are primarily concerned with storing the relationships among sub-patterns. Regularity detectors are concerned with establishing the "meaning" or category associated with a single PE response. Regularity detectors are used when feature discovery is the essential task at hand.

2.3.1 Associative Learning

Associative learning may be broken down into two subcases, pattern-association, and auto-association. A pattern association learning paradigm is one in which the goal is to build up an association between patterns defined over one subset of the PE's and other patterns defined over a second subset of PE's. The goal is to find a set of connections so that whenever a particular pattern reappears on the first set of PE's, the associated pattern will appear on the second set. In this case, there is usually a teaching input to the second set of PE's during training indicating the desired pattern association. An auto-association learning paradigm is one in which an input pattern is associated with itself, or a noisy or incomplete version of itself. The goal in this case is pattern completion. Whenever a portion of the input pattern is presented, the remainder of the pattern is to be filled in or completed. This technique is similar to simple pattern association, except that the input pattern plays both the role of the teaching input and the pattern to be associated. It can be seen that auto-association is a special case of pattern association. Fig 2.5 illustrates the two types of learning paradigms. Figure 2.5a shows the basic structure of pattern association. There are two distinct groups of PE's: a set of input PE's and a set of output PE's. Each input PE connects with output PE's and each output PE receives inputs from input PE's. During training, patterns are presented to both the input and output PE's. The weights connecting the input to the output PE's are modified during this period. During recall, patterns are presented to the input PE's and the response on the output PE's is measured. The input patterns and the corresponding output patterns should reflect the desired input/output pattern relationship. Fig 2.5b shows the connectivity matrix for the pattern associator. The only modifiable connections are from the input PE's to the output PE's, all other connection weights are fixed at zero.

Fig 2.5c illustrates the basic architecture of an auto-associative network. All PE's are both input PE's and output PE's. The figure shows a group of 4 PE's feeding back on itself through modifiable connections. Note that each PE feeds back on itself as well as to each of its neighbors. Fig 2.5d shows the connectivity matrix for the auto-associator. All PE's connect to all other PE's. The weights are modifiable. In the case of auto-association, potentially modifiable connections exist from every PE to every other PE. In the case of pattern association, however, the PE's are

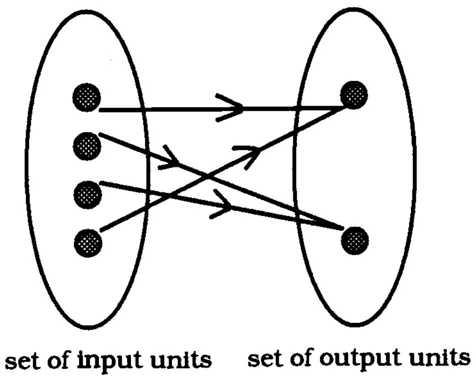


Fig. 2.5a Pattern Association

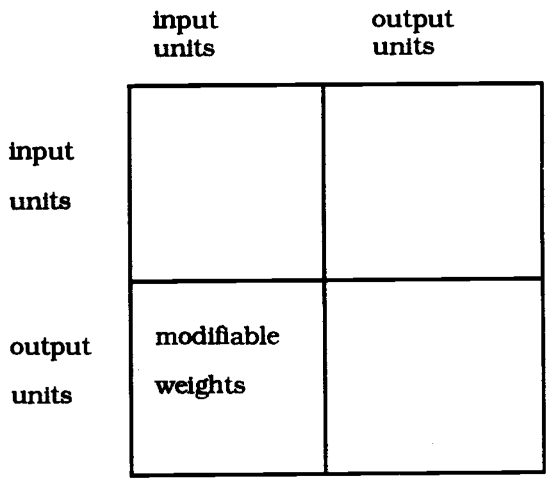


Fig. 2.5b Connection Matrix for Pattern Association

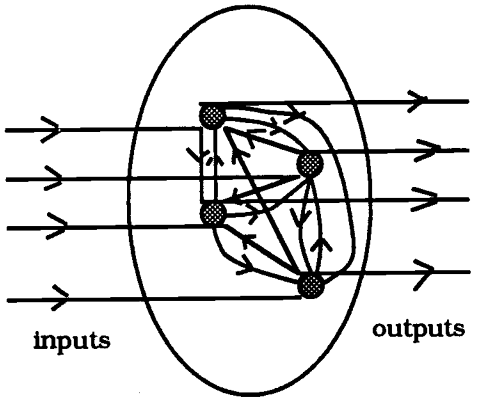


Fig. 2.5c Auto Association

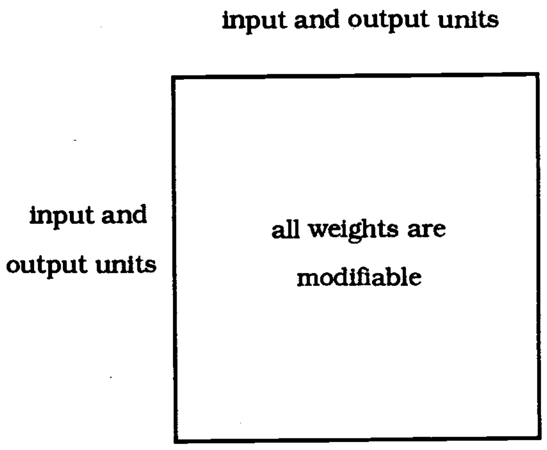


Fig. 2.5d Connection Matrix for Auto Association

Fig. 2.5 Pattern Association and Auto Association

broken into two subgroups, one representing the input patterns and another representing the output patterns(that is teaching input patterns). The only modifiable connections are those from the input PE's to the output PE's receiving the teaching input.

2.3.2. Regularity Discovery

In the case of regularity detectors, a teaching input is not explicitly provided; instead, the teaching function is performed by the PE's themselves. The form of internal teaching function and the nature of the input patterns determine what features the PE will learn to respond to. This activity constitutes unsupervised learning. When a stimulus pattern is presented with some probability P, the system is supposed to discover statistically salient features of the input population. Unlike the associative pattern learning paradigm, there is no a prior set of categories into which the patterns are to be classified; rather, the system must develop its own featural representation of the input stimuli which captures the most salient features of the population of input patterns. The units themselves subdivide the input set into several well-defined classes. Learning via this approach has recently been found to occur in the human olfactory (smell) system.

3 DESIGN AND IMPLEMENTATION

There are many approaches to the character recognition problem. One of the most common way is to use a preprocessor called an invariance network. The output of the invariance network is independent of the translation and rotation of the input patterns to the invariance network. The output of the invariance network is fed into a Back-Propagation network to create standard patterns associated with input patterns. The problem with this approach is that many processing elements are required in the invariance network. Another approach to the problem is to design a multi-layer Back-Propagation network having special connections from the input layer to the hidden layer. This is the famous "T-C problem"[16]. The problem with this solution is that when you have a large number of patterns to classify, it is very difficult to establish the correct connections between the input layer and the hidden layer. The general problem is not as simple as the "T-C" problem in which you can clearly see the solution. Several approaches to this problem were considered. The approach used in this thesis proved to be the most effective.

At the beginning of the research effort, a four layers(two hidden layers) Back-Propagation network was used trying to map the entire input pattern set into three categories. These categories were the standard patterns for the characters "T", "C", and "L". The network included 36 processing elements in its input layer. Each input processing element was connected to one of the pixels on the input grid in the appropriate order. Thirty-six processing elements were used for the first and second hidden layers. In the output layer, we used 25 processing elements, each connected to one of pixels on the output grid in order. A sigmoid function was used as the non-linear transfer function for each processing element. We trained the network using the Back-propagation learning rule. The training patterns were presented to the neural network 5000 times. While training the neural network, the error decreased until it reached an unsatisfactory local minimum, i.e. the whole neural network could only output a few of desired the output patterns. We tried changing the random weights to help the neural network converge, and changing the number of times the training patterns were presented to the neural network. These approaches did not solve the local minimum problem. During the experiment, we found some interesting properties of the network. For a processing element in the output layer, if the desired output for this processing element had more "1"s than "0"s in all of the training patterns, that processing element tended to become stuck in a "1", it would never go to a "0". For a processing element that had more "0"s than "1"s in the training patterns, it tended to become stuck in a "0", and the state would never go to a "1". The three desired output patterns for the network are illustrated in the Fig 3.1. We can see that on the output grid, the upper most row has "1"s for pattern"T" and pattern "C", and has "0"s in the pattern "L". The

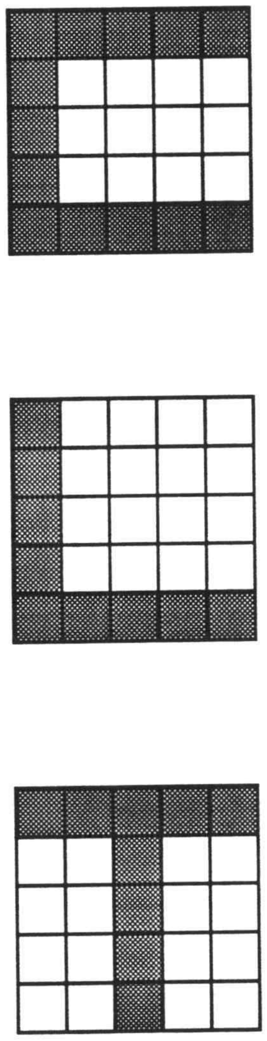


Fig. 3.1 Standard Output Patterns

upper most row of the trained network tended to become stuck in "1"s. The same thing happened to the left most column, and the bottom row. For the middle 3 processing elements in column 3 of the output grid, they have "1"s in pattern"T" but "0"s in patterns"C" and "L", so after training these processing elements tended to become "0"s.

We presented these training patterns to the Back-Propagation network. The result of presenting these patterns was that the neural network produced the pattern"C". We also tried some very simple mapping experiments involving the implementation of the exclusive OR, which worked well. Through these experiments of mapping we found that for simple input-output mapping, the Back-Propagation learning algorithm works well. For very complex input-output mapping structure, it may fail.

We also tried a Hopfield network to implement the character recognition system. The network had 36 processing elements in each layers. After training this network, we presented translated and rotated input patterns to the trained network. As we expected, the trained network could only recall some of the patterns. The problem with this neural network is that it's capacity was too small. Experiments showed by Hopfield that the number of pattern it can store is less than 0.15 times of the processing elements in the input layer[8]. So for a neural network with 36 processing elements it was only possible store 5 patterns. We need to use approximately 300 processing elements to recognize 48 exemplars. There is no way that the Hopfield network of this size could classify those rotated, translated input patterns.

3.1 System Overview

The objective of this thesis is to design a neural network-based English text classifier system. The input set to the neural network will be patterns representing the letters L, C, and T. Dark areas in the patterns will be represented as logical "1" values, and light areas in the patters will be represented as logical "0"'s. We will allow rotation and translation of these patterns within the field of view. This is illustrated in Fig. 3.2. We will therefore expose the network to many views of the same pattern. In addition noise will be added to these patterns. The neural networks under consideration will be able to recognize the characters which are presented to their inputs under the above specified conditions and produce outputs indicative of the characters thus applied. This activity is illustrated in Fig. 3.3. Two different types of neural network classifiers will be examined in the thesis; a classifier which employs the back-propagation network algorithm and one which employs a Hamming network algorithm. The merits of each of these algorithms will be presented.

A grid system is used in characterizing the pixel levels (dark or light) for all input patterns. The grid is composed of a 6 by 6 pixel array. All input patterns can be created using this grid. The input pixels may take on the value +1 (logical "1") for black pixels

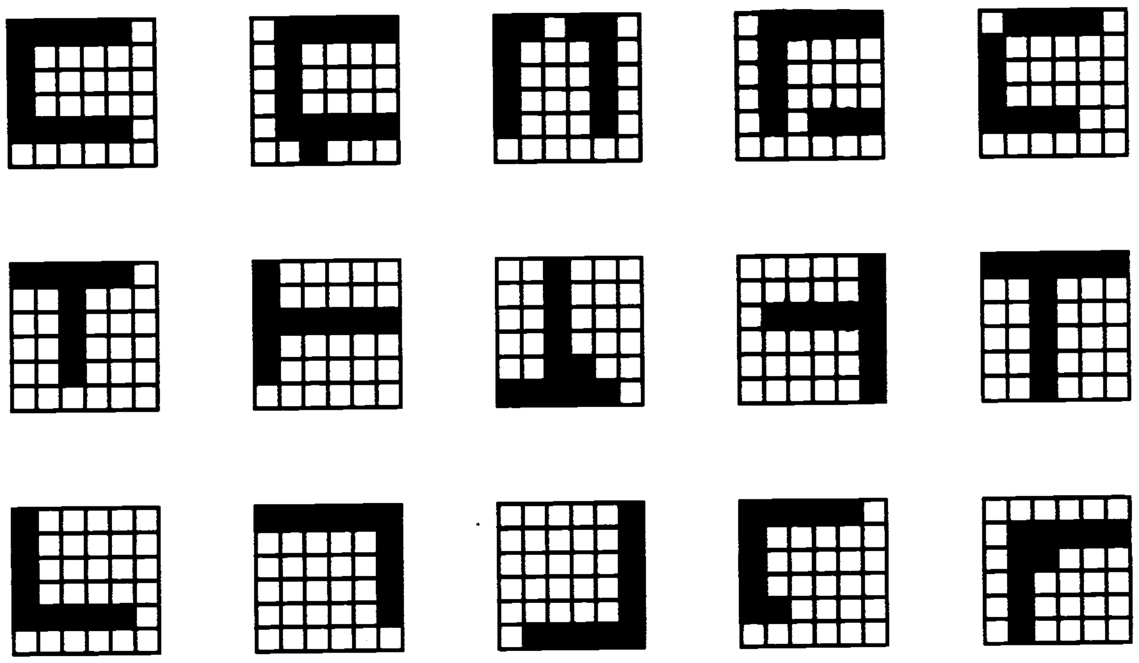


Fig 3.2 Noisy and Rotated Input Patterns

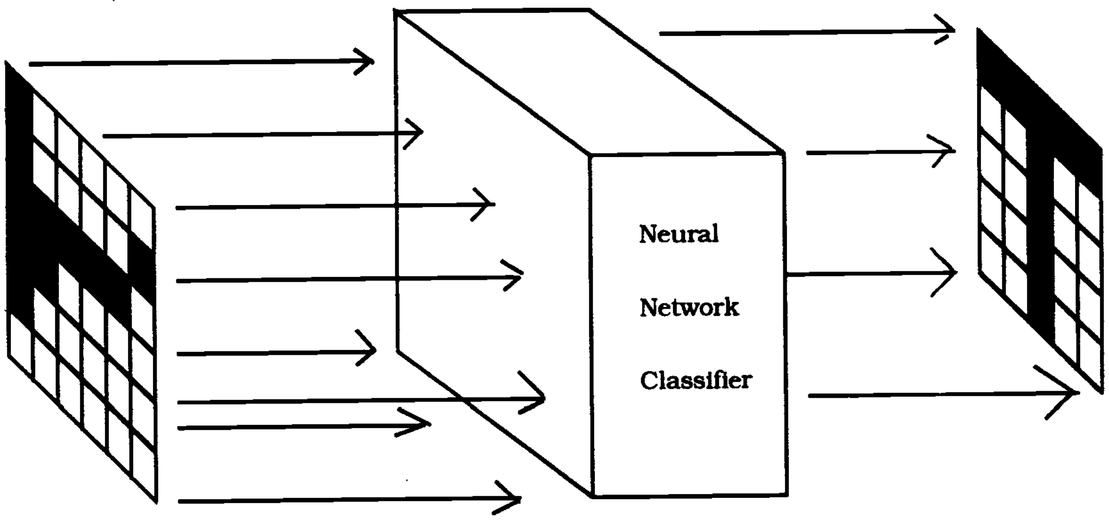


Fig. 3.3 Structure of Character Recognition System

and -1 (logical "0") for white pixels. Each pixel value is applied to one of the processing elements in the input layer. The order of application is illustrated in Fig. 3.4.

3.2 Design Environment

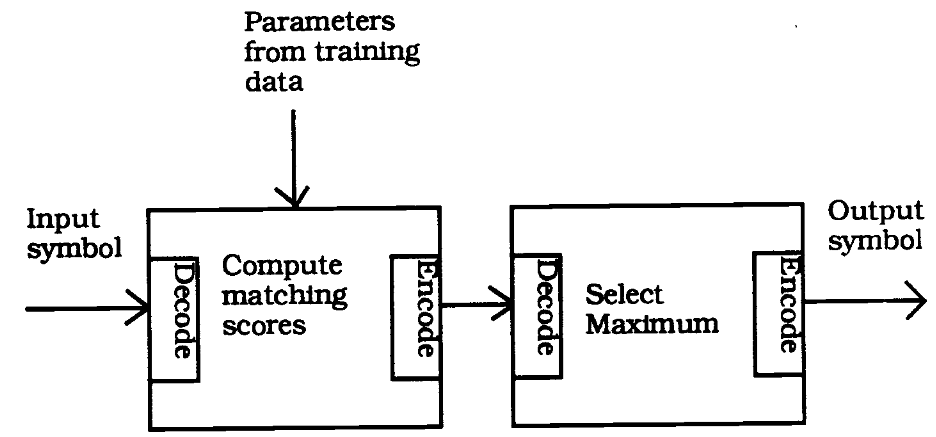
The neural networks described in this thesis were designed with the help of a neural network simulation package. The simulator, called "NeuralWorks" was developed by Neural Ware Inc. It is used by people who wish to design, develop, and test neural networks for specific applications. The software currently runs on IBM PC's, MacIntoshes, and SUN workstations, and the NCUBE parallel processor. The applications may be run, after development, on the NeuralWorks simulator directly, or ported to an external high-performance computer. Many different kinds of neural network can be designed with help of NeuralWorks. The simulator allows you to specify neural network topology, the characteristics of each processing element, to set initial connection strengths or weights, and to specify the learning rule to be utilized. We can also train neural networks via supervised and unsupervised learning techniques, using NeuralWorks. Finally the trained neural networks can be tested to determine if the network is functioning correctly.

3.3 Neural Networks and Traditional Classifiers

Block diagrams of traditional and neural network-based classifiers are presented in Fig. 3.5. Both types of classifiers

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Fig. 3.4 Input Data Grid



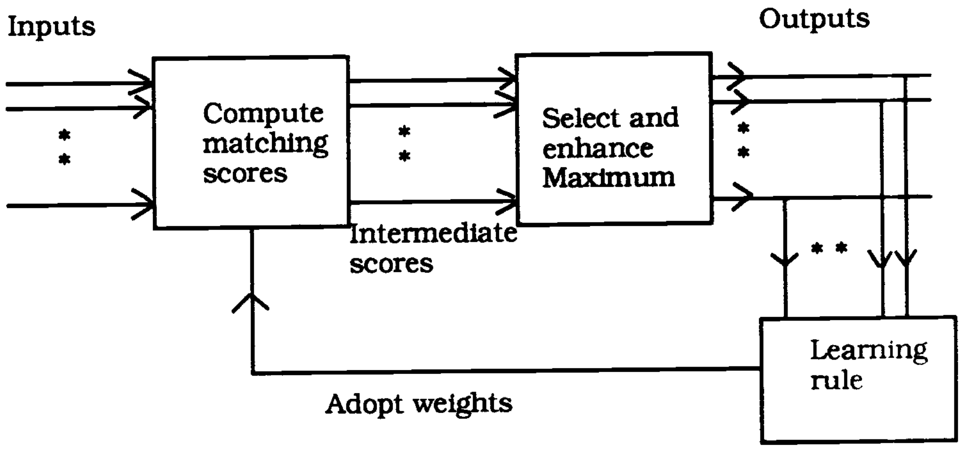


Fig. 3.5 Comparision of Traditional and Neural Network Classifiers

determine which of several possible classes is most representative of an unknown input pattern.

The traditional classifier contains two stages. The first stage computes how well the inputs match the patterns stored within the classifier. The second stage elements select among first stage outputs having the maximum performance in their own class. Inputs to the first stage are symbols representing values of the input elements. These input symbols are entered sequentially and decoded from external symbolic form into an internal representation useful for performing arithmetic and symbolic operations. A classification algorithm computes a matching score for each of the classes which indicates how close the input pattern matches the exemplar or idealized pattern for each class. This exemplar pattern is that pattern which is most representative of each class. Matching scores are coded into symbolic representations and passed sequentially to the second stage of the classifier. Here they are decoded and the class with maximum score is selected. A symbol representing that class is then sent out to complete the classification.

A generalized neural network-based classifier is shown in the Fig. 3.5. In this classifier input values are fed in parallel to the first stage via input connections. Each input connection receives an input value which may take on one of two possible binary values, or the input may vary continuously over a large range of values. The first stage computes a matching score and passes these scores to the next stage. In the second stage, these values are evaluated and the maximum values enhanced. The final stage has one output for each of the output classes. After classification is complete, only that output corresponding to the most likely class will be in a high state; other outputs will be low. Note that in this design, outputs exist for every class and that this multiplicity of outputs must be preserved in further processing stages as long as the classes are considered distinct.

The neural network classifiers in Fig. 3.5 can perform three different tasks. First, they can identify which class best represents an input pattern, where it is assumed that inputs have been corrupted by noise or some other process. This is a classical decision theory problem. Second, the classifier can be used as a content-addressable or associative memory, where the class of the exemplar is the desired output and the input pattern is used to determine which exemplar to produce. A content-addressable memory is very useful when only part of the input pattern is available and the complete pattern is required. It can retrieve the original pattern from partial information. A third task that these classifiers can perform is to vector quantize or cluster the inputs into output clusters. Vector quantizers are used in image and speech transmission systems to reduce the number of bits to be transmitted. In speech and image recognition applications they are used to compress the amount of data that must be processed without losing important information.

3.4 A Framework for System Design

We begin the discussion with a consideration of the input exemplars. The grid we are using for input patterns is composed of 6*6 pixels. We use 5 pixels for each segment of the letters C, L, T. If no noise is mixed with the input patterns, there exist 48 possible input patterns. The number 48 may be calculated as follows: We may move any input pattern upward; downward; right or left, and we can also turn these patterns by 90^o; 180^o; 270^o. The total number of permutations involved in this process is 48. See Fig. 3.6.

The character recognition system is composed of two neural networks in series as shown in the Fig. 3.7, the first neural network (Hamming Net) performs pattern matching activates on the inputs. The connection weights of the Hamming net are determined using the input patterns and the desired output patterns during a "one shot learning" process.

The Hamming net first calculates what is referred to as a "matching score". The matching score indicates the Hamming distance of the input from the 48 possible exemplars. The exemplar with the shortest Hamming distance from the input has the highest matching score. These 48 outputs are then fed into the second neural network, a back-propagation net. Only one of these outputs are active at any time. The back-propagation network then retrieves a standard pattern corresponding to the original 5*5 input pattern, and presents this pattern at its output terminals.

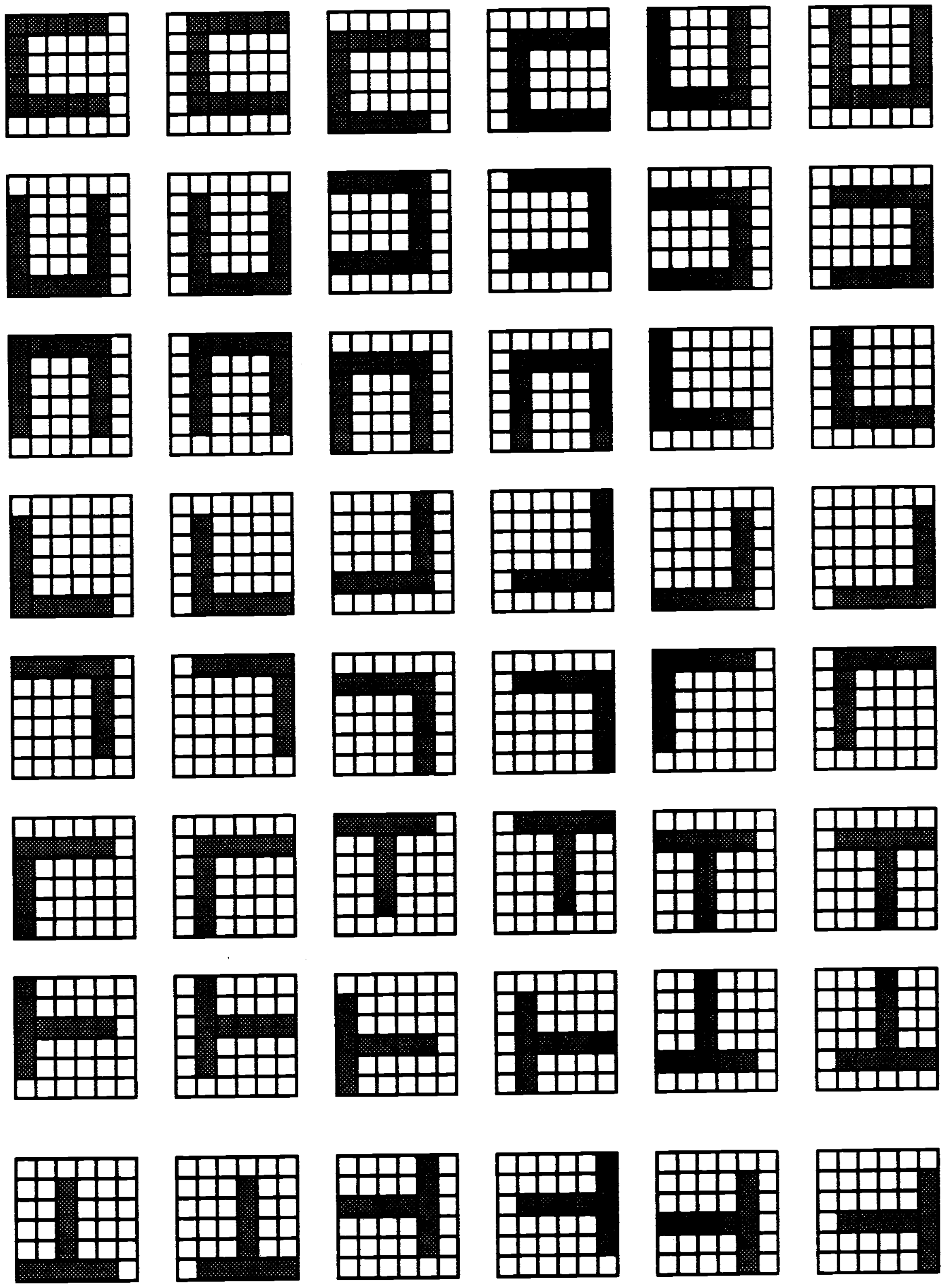


Fig. 3.6 Fourty-eight Exemplars of the Hamming Network

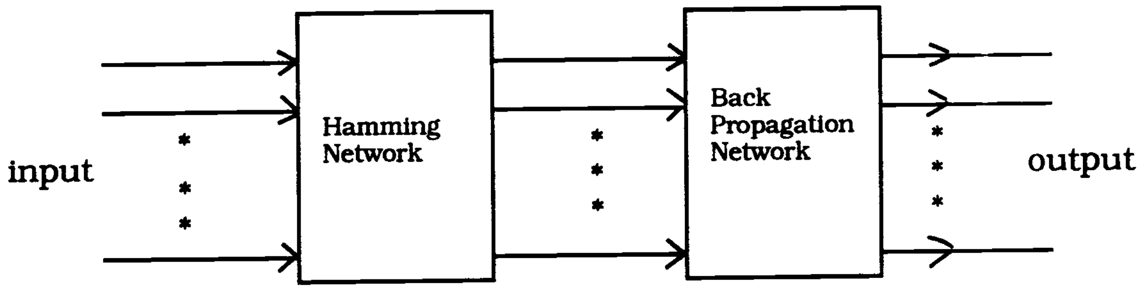


Fig. 3.7 Structure of the Character Recognition System

Following the testing of the character recognition system for basic functionality, we then feed the original input patterns, after having been corrupted by noise, back into the character recognition system. The task for the internal Hamming network is to filter out the noise in the input patterns and correctly produce one of 48 possible input exemplars to be applied to the back-propagation network. When a noisy input pattern is fed to the Hamming net, the network calculates the Hamming distance of the input pattern from the 48 exemplars. Again, whichever exemplar has the shortest Hamming distance from the noisy input pattern will be chosen as the output. The exemplar output of the Hamming net is then fed to the input of the back-propagation network. This network classifies all the exemplars of a particular letter into a specific category and produces a standard pattern output for this letter. The same process applies to all of the letters to be recognized

The output of the system is a 25 bits pattern corresponding to the 5*5 pixel array. The standard output patterns for letter C, L, T are shown in Fig. 3.1. Each time one of the 16 possible exemplars for a specific letter is fed to the back-propagation network, the net will produce a standard output pattern for that letter.

In summary, when a 36 bit pattern corresponding to a particular letter (rotated or translated within the field of view, or mixed with noise) is fed into the neural network classifier system, the system produces a standard pattern for that letter.

3.5 The Design of a Hamming Network Classifier

For the first stage of the neural network classifier, we implemented a Hamming network. A Hamming network implements a minimum error classifier for binary vectors where error is defined using Hamming distance. In a minimum error classifier, classes are defined by means of exemplar vectors. Exemplar vectors contain those +1s and -1s that represent a particular exemplar. Input vectors are then assigned to the class for which the Hamming distance between the exemplar vector and the input vector is minimum. Hamming distance is a way of measuring the distance between two binary vectors and is defined as the number of bits in the input vector which do not match the corresponding bits in the exemplar vector. In this project, we have 48 exemplars. So the Hamming network classifies all inputs into 48 different categories. An unknown input vector is assigned to the category whose exemplar is closest in Hamming distance to the input vector.

3.5.1 Network Topology

A Hamming network consists of two subnets, the first subnet is composed of two layers of processing elements which are fully connected. The second subnet also consists of layer of processing elements. These processing elements function in a competitive manner and perform a "winner-takes-all" function. In this layer, the output of each processing element is fed back to all other processing elements via equal weights as illustrated in Fig. 3.8. This layer is called category layer.

3.5.2 Network Learning Algorithm

The 3-layer Hamming network used in this project has 36 processing elements in its input layer corresponding to the 36 pixels in the input grid. It has 48 processing elements in both the category layer and output layer corresponding to the 48 exemplar vectors. Each processing element in the category layer represents a different classification category as represented by exemplar vector which is encoded in the weights on its incoming connections. These weights are set in the one-shot learning phase as follows:

let

$$x^j = (x^j_1, x^j_2, x^j_3, \ldots , x^j_{36}) \quad j=1, 2, \ldots , 48$$

be the 48 exemplar vectors used to define the categories. We assume that the components x^j_i of x^j take on the values +1 and -1 where +1 represents a black pixel and -1 represents a white pixel. Then the learning phase consists of setting the weights to be

$$w_{ji} = x^j_i / 2.0 \qquad i = 1, 2, \ldots 36, j = 1, 2, \ldots , 48 \qquad [1]$$

$$w_{j0} = 36 / 2.0 \qquad j = 1, 2, \ldots, 48 \qquad [2]$$

where w_{ij} is the weight on the connection from processing element i in the input layer to processing element j in the category layer, and w_{j0} is the weight on the connection from the bias which is a constant input to the processing element j in the category layer.

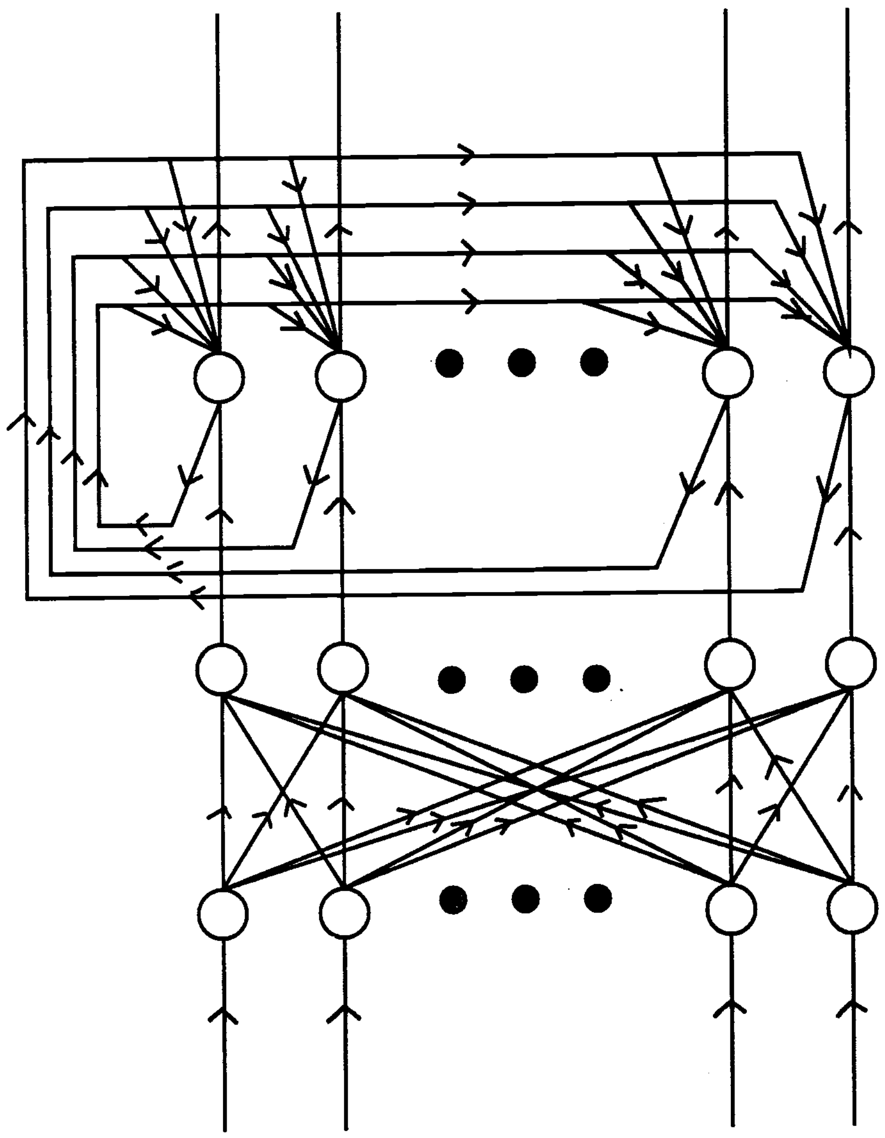


Fig. 3.8 Hamming Network.

The weights are set up this way during one shot learning, and will not be changed during use. It will soon become apparent why the weights are set up in this manner.

During use, when an input vector is presented at the input layer of the system, the input vector is processed through these weighted connections from the input layer to the category layer in the standard way, and combined with the bias term to produce the following input to processing element j in the category layer:

$$I_j = \sum_i \left(w_{ij} x_i\right) \qquad j=1, 2, \ldots, 48 \qquad [3]$$

where

$$x = (x_1, x_2, \ldots, x_{36})$$

is the input vector which, like the exemplar, has components which take on values -1 and +1.

From equations 1, 2, and 3, we have

$$I_j = \tfrac{1}{2} * \left(\sum_i \left(x_i^j x_i\right) + 36 \right) \qquad j=1, 2, \ldots, 48 \qquad [4]$$

Since x_{ji} and x_i only take on the values -1 and +1, equation 4 can be rewritten as

$$I_j = 1/2 * (N_a^j - N_d^j + 36) \qquad j = 1, 2, \ldots, 48 \; [5]$$

where N_a^j is the number of bits where x and x_j agree, and N_d^j is the number of bits where x and x_j disagree. Note that

$$N = N_a^j + N_d^j \qquad j = 1, 2, \ldots 48$$

so equation 5 can be rewritten as

$$I_j = 1/2 * (N_a^j - (N - N_a^j) + N)$$

$= N_a^j$ [6]

$= N - N_d^j \qquad j = 1, 2, \ldots 48$ [7]

Equations 6 and 7 say that the bottom-up connection between the input layer and the category layer calculates N minus the Hamming distance or, equivalently, the number of bits for which the input vector and exemplar vector agree. Every time an input pattern is presented at the input layer, this information is processed through the weighted connections between the input layer and the category layer. The summed input to each processing element j in the category layer is equal to 36 minus the Hamming distance between the input vector and the exemplar j.

The transfer function for processing in the category layer is the perceptron transfer function T_p:

$T_p(I) = I \qquad \text{if } I > 0$ [8]

$T_p(I) = 0 \qquad \text{if } I <= 0$

This processing activity implies that the processing element with the largest initial state will be the one whose exemplar has the smallest hamming distance to the input pattern.

For the second subnet of the Hamming network we use a competitive layer in which the processing element which has the largest initial state will win out, becoming active, and the other processing elements in the same layer will become inactive. This architecture can be implemented in an interactive way through lateral connections in the category layer, as shown in Fig. 3.8. Each processing element in the category layer is laterally connected to every other processing element through a weighted connection having a fixed strength p_{vj} where v and j are v th and j th processing elements in the category layer. These weights are set up as follows

$$p_{vj} = 1.0 \qquad v = j$$

$$p_{vj} = -e \qquad v <> j \qquad 0.0 < e < 1/48$$

in this way, the output of the processing element with the largest initial state will remain relatively high, and the output of other processing elements will be become lower. At the end of the process, just one of the categories will be active (high), that is, having a non-zero output.

We now describe this process mathematically, and we will show how the competition through lateral inhibition evolves. Let us assume that $y_j(t)$ represents the output of processing element j in the category layer at the t th iteration of the competition. From equation 8, the category layer is initialized as:

$$y_j(0) = T_p(I_j)$$

After initialization of the category layer, the input is removed from the category layer, so that the category layer can iterate, without the influence of the input, until stabilization occurs. The output of processing element j at the t th iteration is :

$$y_j(t) = T_p\left(y_j(t-1) - e\sum_{v \neq j} y_v(t-1)\right)$$

This equation shows that, after some number of iterations, the network can converge to a stable state where only one processing element in the layer is active. This processing element is the one which has the largest initial state, and also corresponds to the exemplar that has the shortest Hamming distance to the input vector.

In addition, an output layer exists which is connected to the category layer. We set these connection weights to 1.0. Each processing element in the output layer uses a step function as its transfer function. The processing elements in the output layer therefore only take on the values 1 and 0.

3.5.3 Network Information Capacity and Fault Tolerance

The information capacity of a network is defined as the number of patterns it can store internally. These patterns may be defined in terms of the set of connection weights. We assume there are N processing elements in the input layer of a Hamming network, and there are M processing elements in its category layer. It can store up to 2^N patterns when M= 2^N. But in practical use there are constraints that prevent us from implementing this full set. In the example under consideration, we need to have enough pixels to represent an input pattern. This means we must have enough input processing elements to represent the pattern, but we are not required to consider the full set of possible patterns. Another important fact is that once you store an excessive number of patterns in a Hamming network, it will lose one of the important advantage of neural network, fault tolerance. If, for example, we store 2^N patterns for N input processing elements, and if one bit in a pattern has been changed by noise, the system will consider this pattern as another unique pattern, not the pattern you want. In general it is suggested that all the patterns to be classified have a non-minimum Hamming distance between each other.

3.6 The Design of a Back-Propagation Network Classifier.

For the second part of our neural network classifier, we use a back-propagation network. A back-propagation neural network classifies the output of Hamming network into 3 categories. They are standard pattern for letters 'C' 'L' 'T'. We know that there are 48 exemplars coming from the Hamming network, 16 exemplars for each letter pattern. What it does for the back-propagation network is to classify 16 exemplars for each letter pattern into a some group and output a standard pattern for that letter.

3.6.1 Network Topology

The back-propagation network, illustrated in Fig. 3.9 is a feed forward, multi-layer network. It has an input layer, an output layer, and at least one hidden layer. There are no theoretical limits on the number of hidden layers, but typically one or two are utilized. Research work has indicated that a four layer back-propagation network containing two hidden layers can solve any arbitrarily complex input-output mapping [8]. In this network, each layer is fully connected to the layer adjacent to it. The arrows in Fig. 3.9

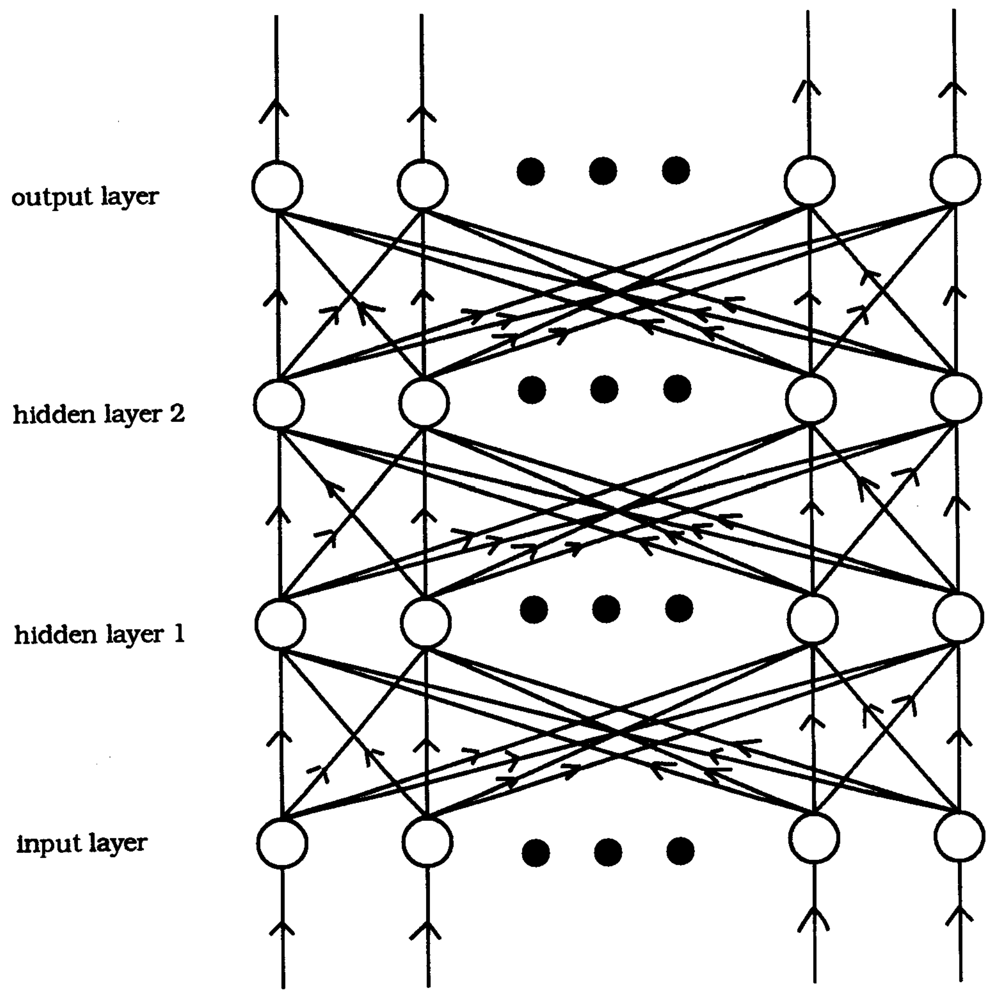


Fig. 3.9 Multi-Layer Back-Propagation Network

indicate the flow of information in the network. Although it can not be proven that the back-propagation algorithm will always converge, they have been proven to be successful for many problems of interest.

The capabilities of multi-layer back-propagation networks stem from the non-linearities used within each processing elements. If these processing elements are linear, then a single-layer network could exactly duplicate those calculations performed by any multi-layer network. The non-linearities help provide feature detection capabilities in what is know as "weight space" that would be otherwise unobtainable.

The back-propagation network designed for this application has three layers. There are 6 linear processing elements in its input layer, 6 non-linear processing elements in its hidden layer, and 25 non-linear processing elements in its output layer. These layers are fully connected. The connection weights in the network are adapted or modified during the learning or training period. Techniques for modifying these weights will be discussed in a later section. The connections between the input layer of the back-propagation network and output layer of the (previous) Hamming network are connected in a special way. The first 16 processing elements of the output layer of the Hamming network are fully connected with the first 2 processing elements of the input layer of the back-propagation network. All the connection weights are set to one. These connections will not be changed during learning.

Identical connections were utilized in rest of the network, as illustrated in Fig. 3.10.

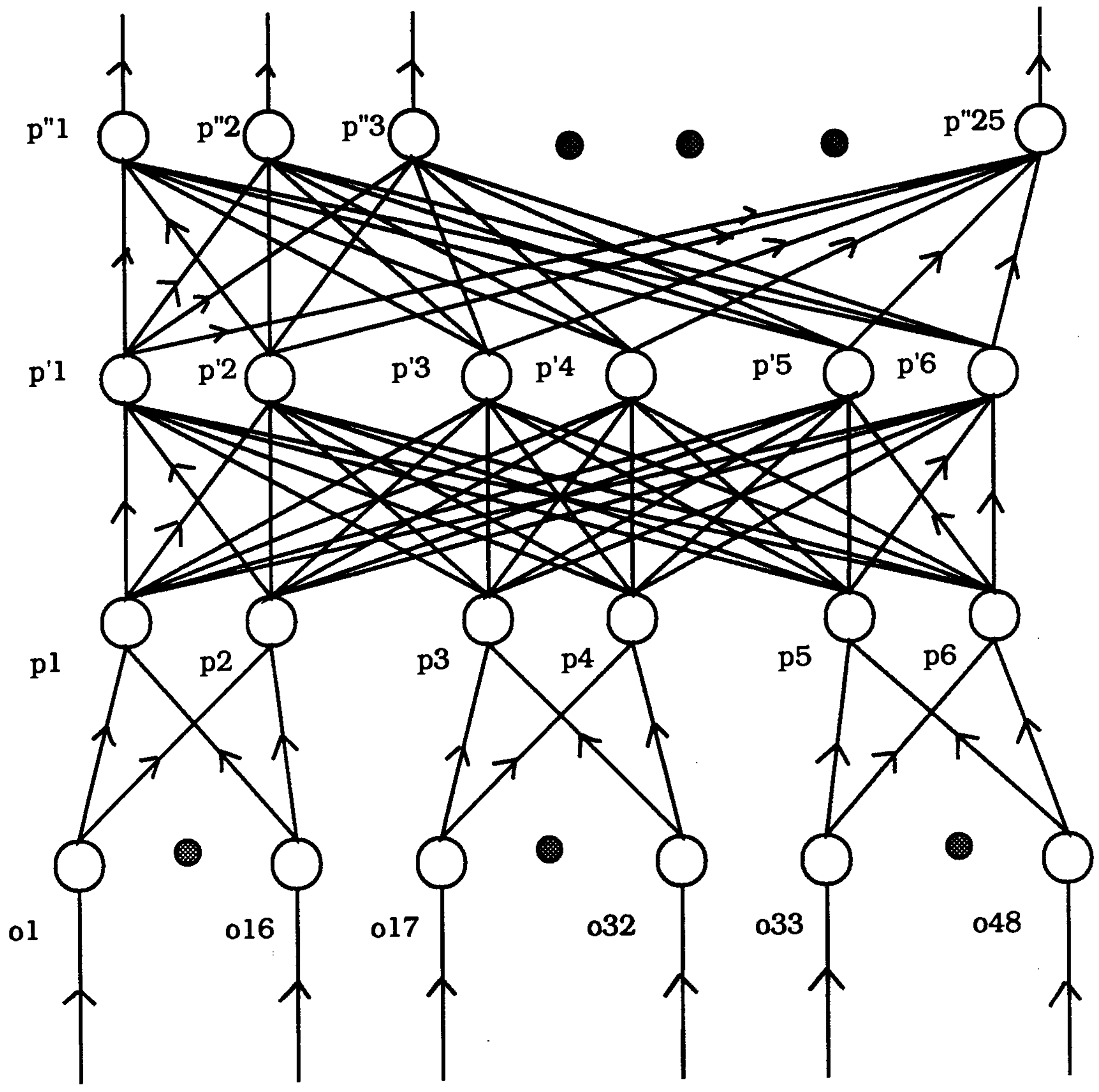
Whenever the weighted sum at the inputs to one of the processing elements in the input layer of the back-propagation network is greater then one, then the processing element output is a one. In this way, whenever one of the processing elements in the output layer of the Hamming network is active high, it will cause two processing elements in the input layer of the back-propagation networks to produce a logical one output.

With this connection scheme, all 16 possible exemplars for a given letter are classified into one unique category. Note that redundancy has been deliberately included in the design of the input layer of the back-propagation network, to achieve some level of fault tolerance.

3.6.2 Network Learning Algorithm

The back-propagation network used in this project has three layers, this is to assure the correct mapping of all inputs into 25 outputs with 3 possible output categories corresponding to the letters C,T, and L. Theory indicates that a network without a hidden layer could only classify those inputs that are linearly separatable [9].

Each output processing element connects to a pixel in the output grid in order to show the standard image of the input letter pattern.



P Represents Processing Elements in the Input Layer of B-P Net
P' Represents Processing Elements in the Hidden Layer of B-P Net
P" Represents Processing Elements in the Output Layer of B-P Net
o Represents Processing Elements in the Output Layer of Hamming Net

Fig. 3.10 Connection Between Back-Propagation and Hamming Network

3.6.2.1 Overview of the Back-Propagation Algorithm

As discussed above, a multi-layer network can solve complex input-output mapping problems. A fundamental question regarding such a network is "How can the weights of the network be adjusted such that we achieve the desired mapping? This constitutes the "credit assignment" problem. The back-propagation training algorithm was the first neural network algorithm to solve this problem and is a generalization of the LMS(least mean square) algorithm. The training algorithm uses gradient search techniques to minimize a cost function equal to the mean square difference between the desired outputs and the actual net outputs for a particular training or input set. It assumes that all processing elements and connections are somewhat responsible for an erroneous output. Responsibility for the error is affixed by propagating the output error backward through the connection to the previous layer. Weights are modified in this layer as a result of the error propagation. This process is repeated until the input layer is reached and all appropriate weights are modified. Repetition of the algorithm occurs until the overall error in the network is reduced below some desired minimum value. This process constitutes a major breakthrough in neural network research.

To avoid confusion, a clear notation is need for describing the learning rule. A standard processing element for a back-propagation network is illustrated in Fig. 3.11. We use a superscript in square brackets to indicate the layer of the network being considered.

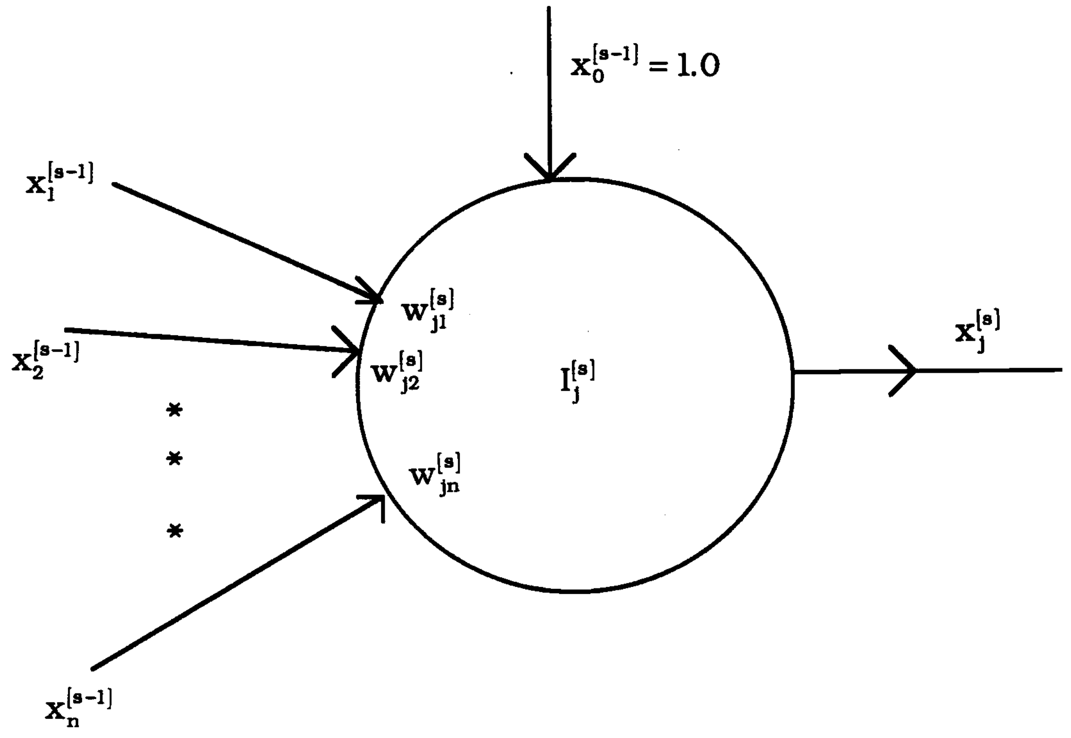


Fig. 3.11 Structure of Processing Element

$x^{[s]}$: current output state of j th neuron in layer s.

$w_{ji}^{[s]}$: weight on connection joining i th neuron in layer (s-1) j th neuron in layer s.

$I_j^{[s]}$: weighted summation of inputs to j th neuron in layer s.

A processing element transfers its input as follows:

$$x_j^{[s]} = f\left(\sum_i \left(w_{ji}^{[s]} * x_i^{[s-1]}\right)\right)$$
$$= f\left(I_j^{[s]}\right)$$

where f can be a any differentiable function.

3.6.2.2 The Global Error Function

The aim of the training process is to minimize the global error of the system by modifying the weights. It is useful to define a global error function, E. This global error function is defined as the collective error at the output layer for a complete training set applied to the inputs to the network. The back-propagation network back propagates this error to modify the connection weights. Suppose a vector i is fed into the input layer of the back-propagation network, and suppose the desired output d is provided by a teacher. Let o denote the actual output produced by the back-propagation network with its current set of weights. Then a measure of the error in achieving that desired output is given by:

$$E = 0.5 * \sum_k \left((d_k - o_k)^2\right)$$

where the subscript k indexes the components of d and o. E defines the global error of the network for a particular pattern.

3.6.2.3 Back-Propagation of the Local Error

Suppose, we have a global error function E. We define a new parameter $e_j^{[s]}$:

$$e_j^{[s]} = -\partial E / \partial I_j^{[s]}$$

This is a critical parameter that is passed back through the layers of the network. The chain rule gives us a relationship between the local error at a particular processing element at level s and all the local error at the level s+1

$$e_j^{[s]} = f'\left(I_j^{[s]}\right) * \sum_k \left(e_k^{[s+1]} * w_{kj}^{[s+1]}\right)$$

As described before f can be any differentiable non-linear function. Here we use a sigmoid function. A sigmoid function is defined as:

$$f(z) = (1.0 + e^{-z})^{-1} \qquad e = 2.718 \ldots$$

so its derivative can be expressed as a simple function of itself :

$$f'(z) = f(z) * (1.0 - f(z))$$

Therefore we have:

$$e_j^{[s]} = x_j^{[s]} * \left(1.0 - x_j^{[s]}\right) * \sum_k \left(e_k^{[s+1]} * w_{kj}^{[s+1]}\right)$$

The summation term in the equation which is used to back-propagate errors is analogous to the summation term which is used to forward propagate the input through the layers to the output layer. So the basic mechanism of back-propagation network is to determine the error between the actual output and the desired output, and then propagate the error back through the network from the output layer to the input layer, modifying the weights in proportion to the error at each level.

3.6.2.4 Minimizing the Global Error by Modifying Weights

Given the current set of weights $w_{ij}^{[s]}$, and a global error function E, it is necessary to determine how to decrease the global error by changing the weights. This can be done by using a gradient descent rule as follows:

$$\Delta w_{ji}^{[s]} = -c_1\left(\partial E / \partial w_{ji}^{[s]}\right)$$

Where c_1 is a learning coefficient. This rule causes the weights to change according to the size and direction of the gradient on the error surface. The partial derivative in the equation can be calculated directly from the local error and the input to that processing element.

$$\begin{aligned}\partial E / \partial w_{ji}^{[s]} &= \left(\partial E / \partial I_j^{[s]}\right) * \left(\partial I_j^{[s]} / \partial w_{ji}^{[s]}\right) \\ &= -e_j^{[s]} * x_i^{[s-1]}\end{aligned}$$

Combining these , we have:

$$\Delta w_{ji}^{[s]} = c_1 * e_j^{[s]} * x_i^{[s-1]}$$

3.6.2.5 Summary of the Back-propagation Learning Algorithm

1. It will be assumed that all processing elements utilize a sigmoidal (nonlinear) transfer function.

2. All weights and offsets for each processing element are to small random values between -0.1 to +0.1.

3 An input vector $x_1, x_2, \ldots x_n$ is presented to the network. The corresponding desired outputs $d_0, d_1, \ldots d_n$. are defined.

4. The actual outputs of the network. $o_1, o_2, \ldots o_n$ are calculated.

5 A recursive algorithm (starting at the output processing elements and working back to the first hidden layer) is utilized to adjust the weights. The weights are adjusted by:

$$w_{ji}(t+1)= w_{ji}(t) + c_1 * e * x_i$$

where:

a $w_{ji}(t)$ is the current weight from a hidden processing element i or from an input processing element to processing element j.

b. $w_{ji}(t+1)$ is the updated weight of the same connection. c_1 is the gain term or learning coefficient. x_i is the input to processing element j or the output of processing element i.

c. e is the error term for processing element j. If processing element j is an output processing element, then :

$$e_j= o_j(1- o_j)(d_j- o_j),$$

d. o_j is the actual output of processing element j of output layer, d_j is the desired output for processing element j of the output layer. If processing element j is an internal hidden processing element, then:

$$e_j = x_j(1-x_j)\sum_i e_i w_{ji}$$

where i is over all processing elements in the layer above processing element j. We can see from here that the error for an internal hidden processing element can be calculated from all the error terms in the above layer, as they are back-propagated through the connection weights.

6. The process is repeated until the desired error measure is achieved.

3.6.2.6 Back-Propagation Momentum Term

One of the important aspects of learning in back-propagation networks is to set up a proper learning rate. If the learning rate is too high, the error may oscillate wildly and never converge to an acceptably small value. If the learning rate is too low, the network will require excessive time to converge. The learning rate is determined by the learning coefficient c_1. If it is assumed that the error surface is locally linear, this keeps learning coefficient small. Small learning coefficients can lead to very slow learning process. In order to improve the time performance of the algorithm and still maintain stability, a momentum term is often introduced into the equation for learning. The learning equation is modified by adding a portion of previous weight.

$$w_{ji}(t+1) = w_{ji}(t) + c_1 * e_j * x_i + c_2(w_{ji}(t) - w_{ji}(t-1))$$

where $0 < c_2 < 1$

This makes the learning process faster. In our simulations, c_1 was set to 0.9, and c_2 was set to 0.6. The resulting simulation had acceptable performance.

3.7 Simulation Results

The complete neural network classifier was simulated using NeuralWorks. We first defined the network topology using NeuralWorks, set up the parameters for each processing element, and defined the required learning rules. It was then necessary to train the system, or allow the system to learn about the input data.

The Hamming Network used the one shot learning algorithm described earlier. After learning is complete, no parameters in the Hamming Net are allowed to change. The input learning file for the Hamming Net is shown in Appendix A. During recall, the input pattern is applied to the trained network. This input pattern consists of the character under consideration after translation, rotation, and/or corruption by noise. When an input pattern is fed to the Hamming Net, the network will output the exemplar that has the least Hamming distance from the input pattern. Test results associated with this process are shown in Appendix B.

The second network in the system, a back-propagation network, was trained using 9 different input patterns. The desired outputs from the net were also presented at the same time. The patterns were applied to the network input repetitively during training. When the output patterns showed no errors, training was stopped, and convergence was achieved. Approximately 500 training cycles were required to achieve zero output errors. The learning curve for the back-propagation network is shown in Fig. 3.12. The actual outputs from the Hamming network were then applied to the back-propagation network to test the network. The correct output pattern was obtained from the back-propagation network in all cases. Test results for the back-propagation network are illustrated in Appendix C.

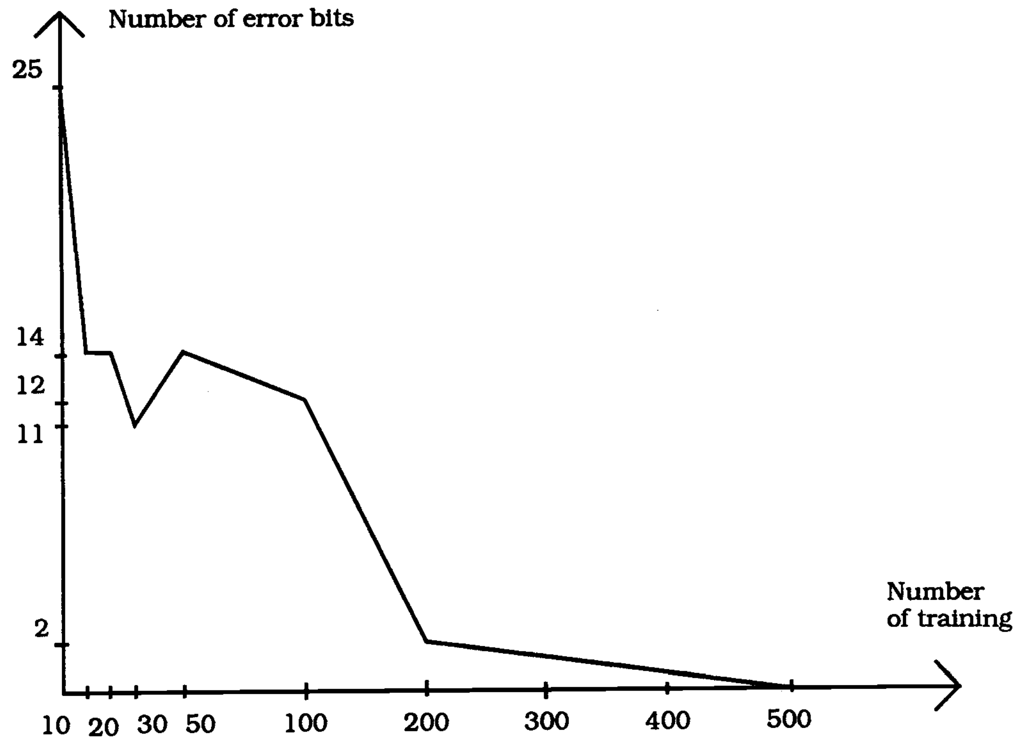


Fig. 3.12 Learning Curve for Back-Propagation Network

4 CONCLUSIONS

4.1 Concluding Remarks

We have designed a neural network classification system specialized to perform character recognition. The neural network classifier implements quite different algorithms comparing with the ordinary pattern recognition algorithms. The system consists of 169 processing elements distributed in 6 layers, with two different learning algorithms implemented in the two internal neural networks. This neural network classifier was functionally simulated using NeuralWorks. The results are quite encouraging. The system worked quite well. It also provided a limited amount of fault tolerance.

Although we did not physically implement the neural network classifier in hardware, it is believed that its processing speed exceeds conventional hardware or software implementations for pattern recognition. The reason for this is that a high degree of parallelism is incorporated in the algorithms which, when reflected in a hardware realization, should provide very high performance.

This project also indicates that neural networks may well have great potential in the fields of speech and pattern recognition and in other area where a large amount of information need to be processed in parallel.

4.2 Future Considerations

The work started in this project should be continued. Specifically, analog VLSI implementation technology should be used to implement a classifier such as this on a silicon chip. Secondly, we need to investigate more powerful and flexible learning rules instead of the elementary back-propagation learning rule for the training of multi-layer neural networks. Although the back-propagation learning rule is considered successful in the research community, there is no guarantee that it will always lead to a correct input-output mapping. Some time the learning rule can lead a neural network into a local minimum in weight space instead of of a global minimum. This may cause the neural network to provide incorrect or at best, suboptimal results.

Finally, it is obvious that more powerful neural network simulators capable of running on parallel machines will be necessary in the future for similar research.

BIBLIOGRAPHY

[1] "An Introduction to Neural Computing", NeuralWorks, P13-P28 NeuralWare, Inc 1988.

[2] W.S. McCulloch, and W. Pitts, "A Logic Calculus of the Ideas Imminent in Nervous Activity," Bulletin of Mathematical Biophysics, 5, 115-133, 1943.

[3] "An Introduction to Neural Computing", NeuralWorks, P347-P366 NeuralWare, Inc 1988.

[4] "An Introduction to Neural Computing", NeuralWorks, P170-P172 NeuralWare, Inc 1988.

[5] B. Widrow, and M.E. Hoff, "Adaptive Switching Circuits", 1960 IRE WESCON Conv. Record, Part 4, 96-104, August 1960.

[6] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA, Vol. 79, 2554-2558, April 1982.

[7] "An Introduction to Neural Computing", NeuralWorks, P438-P465 NeuralWare, Inc 1988.

[8] R. P. Lippmann, "An Introduction to Computing with Neural Nets" , IEEE ASSP Magazine April, 1987.

[9] B. Widrow, R. Winter, " Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition" IEEE, Computer Magazine, March, 1988.

[10] M. Hosokawa, " A new approach for pettern recognition with scramblers", International Joint Conference on Neural Network, 1989, I183

[11] W. E. Weideman, " A comparision of nearest neighbor classifier and a neural network for numberic handprint character recognition", International Joint Conference on Neural Network, 1989, I117

[12] K. Yamada, " Handwritten numberal recognition by a multi-layer neural network", International Joint Conference on Neural Network, 1989, II 259

[13] H. Y. Lee, " Handwritten letter recognition with neural network",International Joint Conference on Neural Network, 1989, II 618

[14] Wuhan Univ.," A model for chinese word recognition in neural network", International Joint Conference on Neural Network, 1989, II 619

[15] D. E. Rumelhart, J. L. McClelland "Parallel Distributed Processing", MIT Press, 1987, P77-P110

APPENDICES

APPENDIX A
TRAINING FILE FOR THE HAMMING NETWORK

This is the training file for hamming net. It has 6*6 inputs and has 48 outputs. In the file, i represents training input, d represents desired output, "1." represents a logical"1", and "-1." represents a logical"0".

```
i 1. 1. 1. 1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
-1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
d 1.
i -1. 1. 1. 1. 1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
-1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1.
d 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
-1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
d 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
-1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
d 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1.
1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
d 0. 0. 0. 0. 1.
i -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 1.
```

```
i -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1.
1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
d 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1.
-1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
-1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
-1. -1. -1. -1. -1. 1. -1. -1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. -1. -1. 1. -1. -1. -1. 1. 1. 1. 1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1.
1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

```
i -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1.
1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1.
-1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1.
1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1.
-1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1.
```

```
-1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1.
1. -1. 1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1.
i -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1.
-1. 1. -1. 1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1.
i 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1.
1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
```

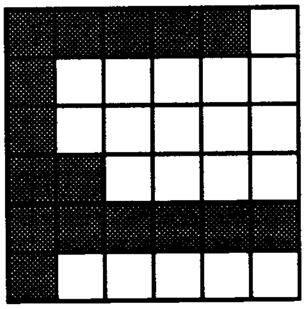
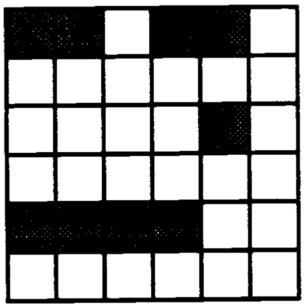
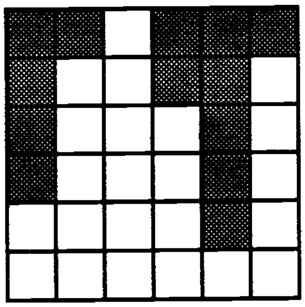
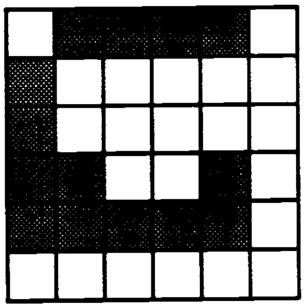
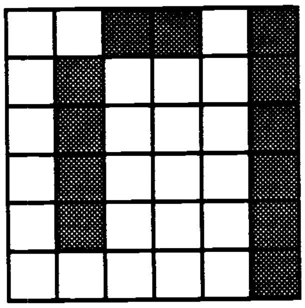
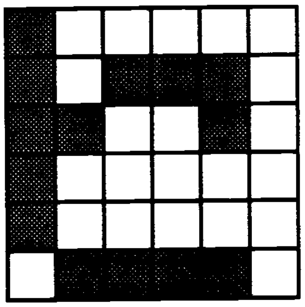
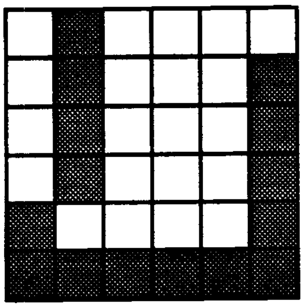
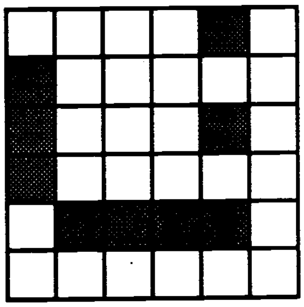
```
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1.
-1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1.
i -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1. -1. 1. -1. 1. -1. -1.
-1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1.
i 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1.
i -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
-1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
-1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1.
```

```
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. 1. 1. 1. 1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1.
-1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1.
1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1.
```

```
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. -1. -1. -1. -1. 1. -1. 1. 1. 1. 1. 1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1.
-1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1. -1.
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
i -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1. -1. -1. 1. -1. -1. -1.
-1. -1. 1. -1. 1. 1. 1. 1. 1. -1. -1. -1. -1. -1. -1.
```

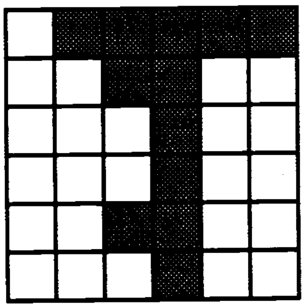
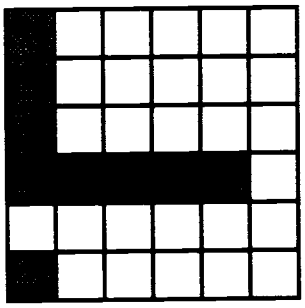
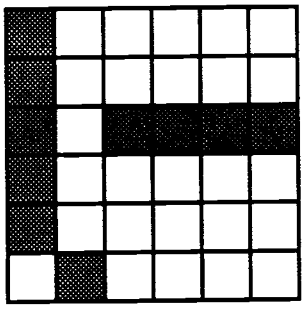
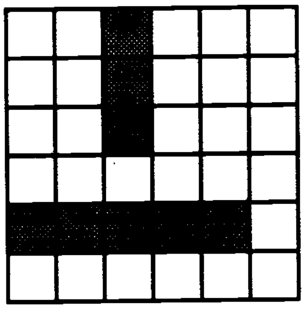
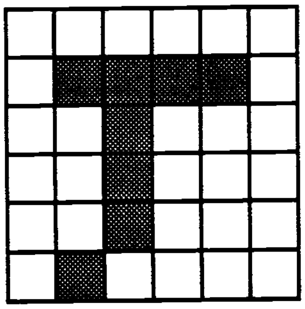
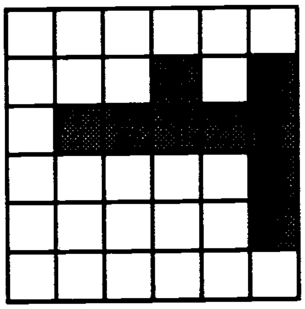
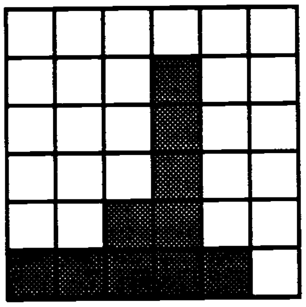
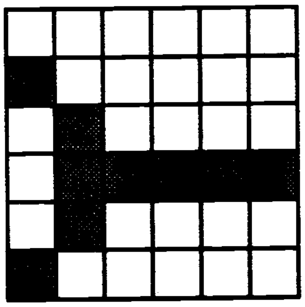
d 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.

APPENDIX B1
TEST RESULTS FOR THE HAMMING NETWORK

Input Patterns	Output	Input Patterns	Output
	'C'/0		'C'/180
	'C'/90		'C'/0
	'C'/90		'C'/0
	'C'/270		'C'/270

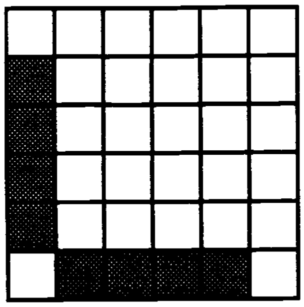
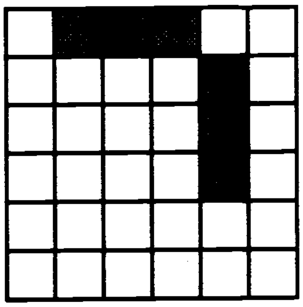
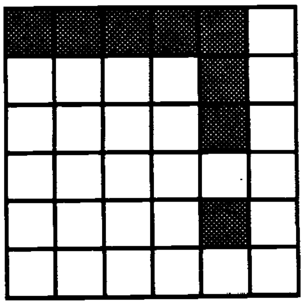
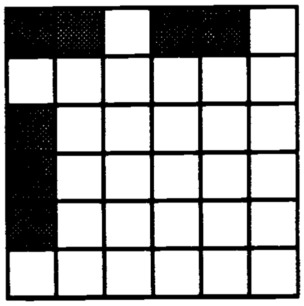
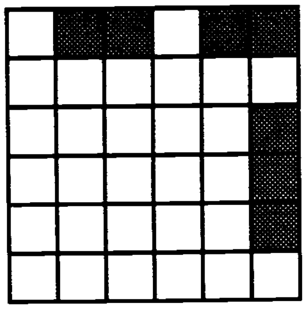
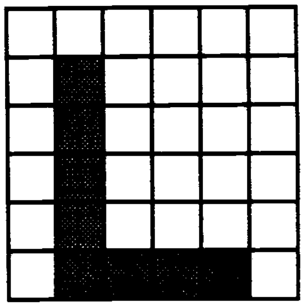
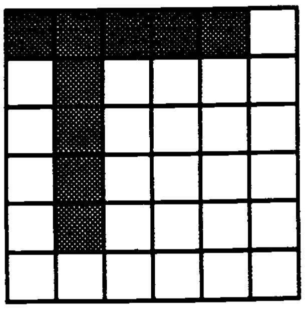
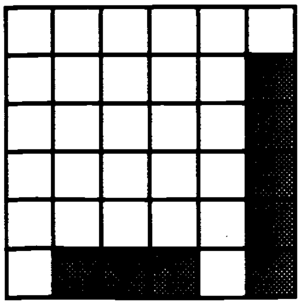
'C'/0 Indicates Letter 'C' Without Rotation
'C'/90 Indicates Letter 'C' With 90 Degree Rotation
'C'/180 Indicates Letter 'C' With 180 Degree Rotation
'C'/270 Indicates Letter 'C' With 270 Degree Rotation

APPENDIX B2
TEST RESULTS FOR THE HAMMING NETWORK

Input Patterns	Output	Input Patterns	Output
	'T'/0		'T'/270
	'T'/270		'T'/180
	'T'/0		'T'/90
	'T'/180		'T'/270

'T'/0 Indicates Letter 'T' Without Rotation
'T'/90 Indicates Letter 'T' With 90 Degree Rotation
'T'/180 Indicates Letter 'T' With 180 Degree Rotation
'T'/270 Indicates Letter 'T' Wth 270 Degree Rotation

APPENDIX B3
TEST RESULTS FOR THE HAMMING NETWORK

Input Patterns	Output	Input Patterns	Output
	'L'/0		'L'/180
	'L'/180		'L'/90
	'L'/90		'L'/0
	'L'/90		'L'/270

'L'/0 Indicates Letter 'L' Without Rotation
'L'/90 Indicates Letter 'L' With 90 Degree Rotation
'L'/180 Indicates Letter 'L' With 180 Degree Rotation
'L'/270 Indicates Letter'L' With 270 Degree Rotation

APPENDIX C

TEST RESULT FOR THE BACK-PROPAGATION NETWORK

INPUT VECTOR FOR BACK-PROPAGATION NETWORK

OUTPUT PATTERNS

```
  1 to 16 bit     17 to 32 bit     33 to 48 bit

1000....0000....0000....0000
0100....0000....0000....0000
0010....0000....0000....0000
0001....0000....0000....0000
  *
  *
  *
0000....1000....0000....0000
0000....0100....0000....0000
0000....0010....0000....0000
0000....0001....0000....0000
  *
  *
  *
0000....0000....1000....0000
0000....0000....0100....0000
0000....0000....0010....0000
0000....0000....0001....0000
  *
  *
  *
0000....0000....0000....1000
0000....0000....0000....0100
0000....0000....0000....0010
0000....0000....0000....0001
```