NEAR EAST UNIVERSITY LIBRARY

NEAR EAST UNIVERSITY

Faculty Of Engineering

Department Of Computer Engineering

NEURAL NETWORK SYSTEM FOR CHARACTER RECOGNITION

Graduating Project
COM 400

Student: Ayfer Tözer

Supervisor : Assoc.Prof.Dr. Rahib Abiyev

Nicosia 2006

ACKNOWLADGEMENTS

Firstly, I would like to thank to my supervisor Mr Rahib Abiyev for his great advise and recomendation for finishing my project properly also, teaching and guiding me in others lectures

I am greatly indepted to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my sisters Funda and Pınar Tözer did for me during my educational life, also I want to say thanks to my brother Mr. Abdurrahman Tözer. And of course, to forget is impossible my Family. I dedicate my project to them.

Althougt, I encountered many problems in writing program. Also ı want to thanks to asistant of my department Boran Şekeroğlu Koray Altunkaya. My sincere thanks to him and others my class friends.

I thank all the staff of the faculty of engineering for giving facilities to practise, teaching and solving problem in my comlete undergraduation program

Finally, I promise to do my best in my life as an bachelor of engineer.

ABSTRACT

In the real life, there are such image recognition problems that need construction of intelligent systems with high accuracy and very quick decision-making mechanism. One of approaches to solve these problems is Neural Networks.

Neural network applications have lately become a common feature with varying degrees of success and usability. Neural networks can provide sufficient and robust solutions to problems where automation is required.

The back-propagation learning algorithm has been implemented and tested on application, character recognition. Experimental results including training, time and recognition accuracy are given.

TABLE OF CONTENTS

List of Figure

List of Tables

INTRODUCTION

Many recognition techniques such as image processing techniques and a neural network (NN) technique have been proposed for the character recognition. Based on several previous research studies, it seems that the NN technique is more suitable and robust for recognition than any other techniques because of its self-organization and generalization abilities. Therefore, the NN technique has been applied to the character recognition system.

This recognition system has been applied to recognize one set of characters with a "Times New Roman" font. System has been trained with a clean set of these characters and tested by trained set and noisy character set which are not trained to the neural network.

The aims of the work presented within this project:

- To investigate Neural Networks algorithms and techniques,
- To provide an intelligent character recognition system
- To design and simulate a neural network for character recognition using Matlab.

This project consists of 4 chapters and a conclusion. First three chapters give an introduction about the background of this work, Neural Networks and Character Recognition Systems and the last chapters explain the work done.

CHAPTER 1
INTRODUCTION TO CHARACTER RECOGNITION SYSTEMS

1.1 Object Recognition

Object recognition consists of locating the positions and possibly orientations and scales of instances of objects in an image. The purpose may also be to assign a class label to a detected object. Our survey of the literature on object recognition using ANNs indicates that in most applications, ANNs have been trained to locate individual objects based direction pixel data. Another less frequently used approach is to map the contents of a window onto a feature space that is provided as input to a neural classifier.

1.1.1 Optical Character Recognition

The recognition of handwritten or printed text by computer is referred to as Optical Character Recognition. When the input device is a digitizer tablet that transmits the signal in real time (as in pen-based computers and personal digital assistants) or includes timing information together with pen position (as in signature capture) we speak of dynamic recognition. When the input device is a still camera or a scanner, which captures the position of digital ink on the page but not the order in which it was laid down, we speak of static or image-based OCR.

Dynamic OCR is an increasingly important modality in Human Computer I interaction, and the difficulties encountered in the process are largely similar to those found in other HCI modalities, in particular, Speech Recognition. The stream of position/pen pressure values output by the digitizer tablet is analogous to the stream of speech signal vectors output by the audio processing front end, and the same kinds of lossy data compression techniques, including cepstral analysis, linear predictive coding, and vector quantization, are widely employed for both.

Static OCR encompasses a range of problems that have no counterpart in the recognition of spoken or signed language, usually collected under the heading of page decomposition or layout analysis. These include both the separation of linguistic material from photos, line drawings, and other non-linguistic information, establishing the local horizontal and vertical axes (deskewing), and the appropriate grouping of titles, headers, footers, and other material set in a font different from the main body of the text. Another OCR-specific problem is that we often find different scripts, such as Kanji and Kana, or Cyrillic and Latin, in the same running text.

While the early experimental OCR systems were often rule-based, by the eighties these have been completely replaced by systems based on statistical, Pattern Recognition. For clearly segmented printed materials such techniques offer virtually error-free OCR for the most important alphabetic systems including variants of the Latin, Greek, Cyrillic, and Hebrew alphabets.

However, when the number of symbols is large, as in the Chinese or Korean writing systems, or the symbols are not separated from one another, as in Arabic or Devanagari print, OCR systems are still far from the error rates of human readers, and the gap between the two is also evident when the quality of the image is compromised e.g. by fax transmission. Until these problems are resolved, OCR can not play the pivotal role in the transmission of cultural heritage to the digital age that it is often assumed to have.

In the recognition of handprint, algorithms with successive segmentation, classification, and identification (language modeling) stages are still in the lead, as shown in the later chapters.

1.2 Character Recognition

It is often useful to have a machine perform pattern recognition. In particular, machines that can read symbols are very cost effective. A machine that reads banking checks can process many more checks than a human being in the same time. This kind of application saves time and money, and eliminates the requirement that a human perform such a repetitive task. The script appcr1 demonstrates how character recognition can be done with a backpropagation network. Problem StatementA network is to be designed and trained to recognize the 26 letters of the alphabet. An imaging system that digitizes each letter centered in the system's field of vision is available. The result is that each letter is represented as a 5 by 7 grid of boolean values. For example, here is the letter A. However, the imaging system is not perfect and the letters may suffer from noise. Perfect classification of ideal input vectors is required, and reasonably accurate classification of noisy vectors. The twenty-six 35-element input vectors are defined in the function prprob as a matrix of input vectors called alphabet. The target vectors are also defined in this file with a variable called targets. Each target vector is a 26-element vector with a 1 in the position of the letter it represents, and 0's everywhere else. For example, the letter A is to be represented by a 1 in the first element (as A is the first letter of the alphabet), and 0's in elements two through twenty-six.

1.3 Pattern Recognition

The act of recognition can be divided into two broad categories: recognizing concrete items and recognizing abstract items. The recognition of concrete items involves the recognition of spatial and temporal items. Examples of spatial items are fingerprints, weather maps, pictures and physical objects. Examples of temporal items are waveforms and signatures. Recognition of abstract items involves the recognition of a solution to a problem, an old conversation or argument, etc. In other words, recognizing items that do not exist physically.

1.3.1 Filtering

Filtering is removing unwanted information or data from input. Depending on the application, the filter algorithm or method will change. For example, consider finger print identification. Each time we scan our fingerprints through a (non-ink) fingerprint device, the scanned output may be different. The difference may be due to a change in contrast or brightness or in the background of the image. There could be some distortion. In order to process the input, we may need only lines in the fingerprints and we may not need the other parts or background of the fingerprint. In order to filter out the unwanted portion of the image and replace it with a white background, we need a filter mechanism. Once the image is filtered through the filter mechanism, we will get standard clean finger prints only with lines, which in turn helps with the process of feature extraction.

1.3.2 Feature extraction

Is a process of studying and deriving useful information from the filtered input patterns. The derived information may be general features, which are evaluated to ease further processing. For example, in image recognition, the extracted features will contain information about gray shade, texture, shape or context of the image. This is the main information used in image processing. The methods of feature extraction and the extracted features are application dependent.

1.3.3 Classification

Is the final stage of the pattern recognition. This is the stage where an automated system declares that the inputted object belongs to a particular category. There are many classification methods in the field. Classification method designs are based on the following concepts.

1.3.4 Member-roster concept

Under this template-matching concept, a set of patterns belonging to a same pattern is stored in a classification system. When an unknown pattern is given as input, it is compared with existing patterns and placed under the matching pattern class.

1.3.5 Common property concept

In this concept, the common properties of patterns are stored in a classification system. When an unknown pattern comes inside, the system checks its extracted common property against the common properties of existing classes and places the pattern/object under a class, which has similar, common properties.

1.3.6 Clustering concept

Here, the patterns of the targeted classes are represented in vectors whose components are real numbers. So, using its clustering properties, we can easily classify the unknown pattern. If the target vectors are far apart in geometrical arrangement, it is easy to classify the unknown patterns. If they are nearby or if there is any overlap in the cluster arrangement, we need more complex algorithms to classify the unknown patterns. One simple algorithm based on the clustering concept is Minimum Distance Classification. This method computes the distance between the unknown pattern and the desired set of known patterns and determines which known pattern is closest to the unknown and, finally, the unknown pattern is placed under the known pattern to which it has minimum distance. This algorithm works well when the target patterns are far apart.

1.4 Neural Network

The network receives the 35 Boolean values as a 35-element input vector. It is then required to identify the letter by responding with a 26-element output vector. The 26 elements of the output vector each represent a letter. To operate correctly, the network should respond with a 1 in the position of the letter being presented to the network. All other values in the output vector should be 0. In addition, the network should be able to handle noise. In practice, the network does not receive a perfect Boolean vector as input. Specifically, the network should make as few mistakes as possible when classifying vectors with noise of mean 0 and standard deviation of 0.2 or less.

1.5 Architecture

The neural network needs 35 inputs and 26 neurons in its output layer to identify the letters. The network is a two-layer log-sigmoid/log-sigmoid network. The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output boolean values. The hidden (first) layer has 10 neurons. This number was picked by guesswork and experience. If the network has trouble learning, then neurons can be added to this layer. The network is trained to output a 1 in the correct position of the output vector and to fill the rest of the output vector with 0's. However, noisy input vectors may result in the network not creating perfect 1's and 0's. After the network is trained the output is passed through the competitive transfer function compet. This makes sure that the output corresponding to the letter most like the noisy input vector takes on a value of 1, and all others have a value of 0. The result of this post-processing is the output that is actually used.

1.6 Initialization of Neural Network

Initial values of back propagation learning algorithm can be seen below.

```
S1 = 100;
[R,Q] = size(P2);
[S2,Q] = size(T);
net = newff(minmax(P2),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

1.7 Training of Neural Network

Final training parameters of neural network can be seen below.

```
net.performFcn = 'sse';
net.trainParam.goal = 0.01;
net.trainParam.show = 20;
net.trainParam.lr = 0.003;
net.trainParam.epochs = 10000;
net.trainParam.mc = 0.15;
[net,tr] = train(net,P2,T);
save('TrainedNetwork.mat','P2','P1','T','net','tr');
```

CHAPTER 2
NEURAL NETWORKS

2.1Overview

This chapter presents an introduction about the general neural networks, development of neural networks, biological neural networks, artificial models and methods of neural networks. Also major components of artificial neurons, connection types, learning laws and comparison of neural networks with traditional computing will be explained.

2.2 Origins of Neural Networks

In the early 1940s, Warren McCulloch (1899-1969, was an American neurophysiologist and cybernetician[1]) and Walter Pitts (1928-late 1960's, was logician who worked in the field of Cognitive Psychology) published a seminar paper titled "A Logical Calculus of the Ideas Immanent in Nervous Activity". In it, they proposed a mathematical model of a neuron, which could perform computations. This artificial neuron, or neurode (some call them neurons), was a simple device, which could receive input from other such devices [1].

The neurode's output was either a 1 or a 0, reflecting the *all-or-none* theory of biological neurons. When the total input reached a certain critical level, the neurode would send its output to other neurodes with which it was connected. This method is called *threshold logic.* In basic propositional logic, something can be either true or false. Since a neurode's state is either a 1 or a 0, it can be represented by a proposition. If you organize simple neurodes into a network, they can combine to form more complex propositions. This theory was so influential that this type of neurode is called the *McCulloch- Pitts neuron.* Some modern neural networks use neurodes, which are essentially extensions of the McCulloch-Pitts neuron.

The concept of neural networks has been around since the early 1950s, but was mostly dormant until the mid 1980s. One of the first neural networks developed was the *perceptron.* Created by a psychologist named Frank Rosenblatt in 1958, *the perceptron* was a very simple system that used interconnected neurodes to analyze data, usually visual patterns. Rosenblatt published a series of papers, which generated a great deal of interest in the perceptron. Many people researched and developed further the perceptron model, even implementing it in hardware. The perceptron was widely and unrealistically praised by researchers. Rosenblatt and other scientists claimed that eventually, with enough complexity and speed, the perceptron would be able to solve almost any problem.

This was far from the truth. In 1969, Marvin Minsky (1927, is an American scientist in the field of Artificial Intelligence) and Seymour Papert (1928, is an MIT[1] mathematician. He is one of the pioneers of AI) published an influential book titled "Perceptrons". In it, they proved several theorems, which showed that the perceptron could never solve a class of simple problems, and hinted at several other serious, fundamental flaws in the model. After "Perceptrons", scientists working on neural network type devices found it almost impossible to receive funding.

2.3 Biological Neuron

The brain is a collection of about 10 billion interconnected neurons [1]. Each neuron is a cell that uses biochemical reactions to receive, process and transmit information. A neuron's dendritic tree is connected to a thousand neighboring neurons. When one of those neurons fire, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation. Spatial summation occurs when several weak signals are converted into a single large one, while temporal summation converts a rapid series of weak pulses from one source into one large signal. The aggregate input is then passed to the soma (cell body). The soma and the enclosed nucleus don't play a significant role in the processing of incoming and outgoing data. Their primary function is to perform the continuous maintenance required to keep the neuron functional. The part of the soma that does concern itself with the signal is the axon hillock. If the aggregate input is greater than the axon hillock's threshold value, then the neuron *fires*, and an output signal is transmitted down the axon. The strength of the output is constant, regardless of whether the input was just above the threshold, or a hundred times as great. The output strength is unaffected by the many divisions in the axon; it reaches each terminal button with the same intensity it had at the axon hillock. This uniformity is critical in an analogue device such as a brain where small errors can snowball, and where error correction is more difficult than in a digital system (Figure 2.1).

Each terminal button is connected to other neurons across a small gap called a synapse [1] (Figure 2.2). The physical and neurochemical characteristics of each synapse determines the strength and polarity of the new input signal. This is where the brain is

the most flexible, and the most vulnerable. Changing the constitution of various neuro-transmitter chemicals can increase or decrease the amount of stimulation that the firing axon imparts on the neighboring dendrite. Altering the neurotransmitters can also change whether the stimulation is excitatory or inhibitory. Many drugs such as alcohol and LSD have dramatic

effects on the production or destruction of these critical chemicals. The infamous nerve gas sarin can kill because it neutralizes a chemical (acetyl cholinesterase) that is normally responsible for the destruction of a neurotransmitter (acetylcholine). This means that once a neuron fires, it keeps on triggering all the neurons in the vicinity. One no longer has control over muscles, and suffocation ensues.

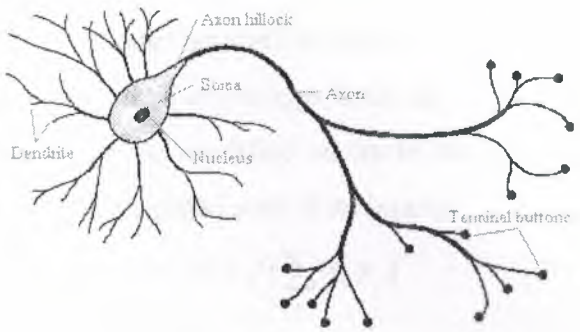


Figure 2.1 Schematic of Biological Neuron

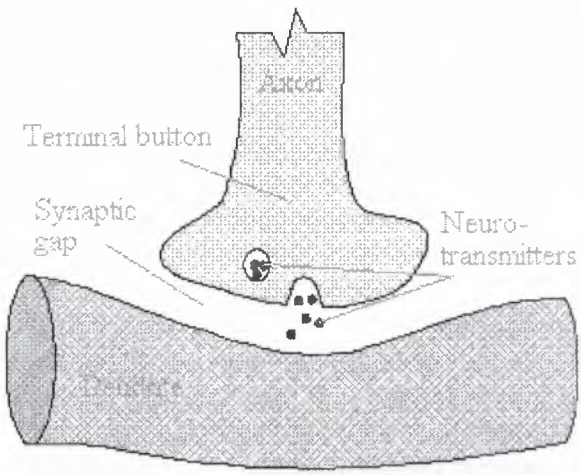


Figure 2.2 The synapse

2.4 Artificial Neuron Models

Computational neurobiologists have constructed very elaborate computer models of neurons in order to run detailed simulations of particular circuits in the brain. As Computer Scientists, we are more interested in the general properties of neural networks; independent of how they are actually "implemented" in the brain. This means that we can use much simpler, abstract "neurons", which (hopefully) capture the essence of neural computation even if they leave out much of the details of how biological neurons work.

People have implemented model neurons in hardware as electronic circuits, often integrated on VLSI chips. Remember though that computers run much faster than brains - we can therefore run fairly large networks of simple model neurons as software simulations in reasonable time. This has obvious advantages over having to use special "neural" computer hardware.

2.4.1 An Artificial Neuron

Basic computational element (model neuron) is often called a node or unit (Figure 2.3). It receives input from some other units, or perhaps from an external source. Each input has an associated weight w, which can be modified so as to model synaptic learning. The unit computes some function f of the weighted sum of its inputs:

$$y_i = f(\sum_j w_{ij} y_j)$$

ts output, in turn, can serve as input to other units.

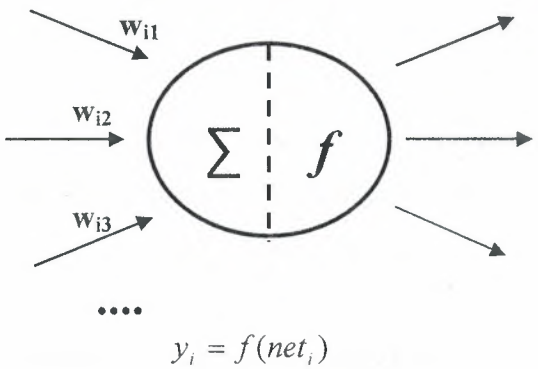


Figure 2.3 A Simple Artificial Neuron

- The weighted sum $\sum_j w_{ij} y_j$ is called the *net input* to unit i, often written net_i.
- Note that w_{ij} refers to the weight from unit j to unit i (not the other way around).
- The function f is the unit's *activation function.* In the simplest case, f is the identity function, and the unit's output is just its net input. This is called a *linear unit.*

2.4.2 Major Components of Artificial Neurons

Major components of an artificial neuron are described as shown below [1]. These components are valid whether the neuron is used for input, output, or is in one of the hidden layers.

2.4.2.1 Weighting Factors

A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input the impact that it needs on the processing element's summation function. These weights perform the same type of function, as do the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response. Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

2.4.2.2 Summation Function

The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as $(i_1, i_2 \ldots i_n)$ and $(w_1, w_2 \ldots w_n)$. The total input signal is the dot, or inner, product of these two vectors. This simplistic summation function is found by multiplying each component of the i vector by the corresponding component of the w vector and then adding up all the products. $Input_1 = i_1 * w_1$, $input_2 = i_2 * w_2$, etc., are added as $input_1 + input_2 + \ldots + input_n$. The result is a single number, not a multi-element vector. Geometrically, the inner product of two vectors can be considered a measure of their similarity. If the vectors point in the same direction, the inner product is maximum; if the vectors point in opposite direction (180 degrees out of phase), their inner product is minimum. The summation function can be more complex than just the simple input and weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to a simple product summing, the summation function can select the minimum, maximum, majority, product, or several normalizing

algorithms. The specific algorithm for combining neural inputs is determined by the chosen network architecture and paradigm.

Some summation functions have an additional process applied to the result before it is passed on to the transfer function. This process is sometimes called the *activation function*. The purpose of utilizing an activation function is to allow the summation output to vary with respect to time. Activation functions currently are pretty much confined to research. Most of the current network implementations use an "identity" activation function, which is equivalent to not having one. Additionally, such a function is likely to be a component of the network as a whole rather than of each individual processing element component.

2.4.2.3 Transfer Function

The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal. If the sum of the input and weight products is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant. The threshold, or transfer function, is generally non-linear. Linear (straight-line)

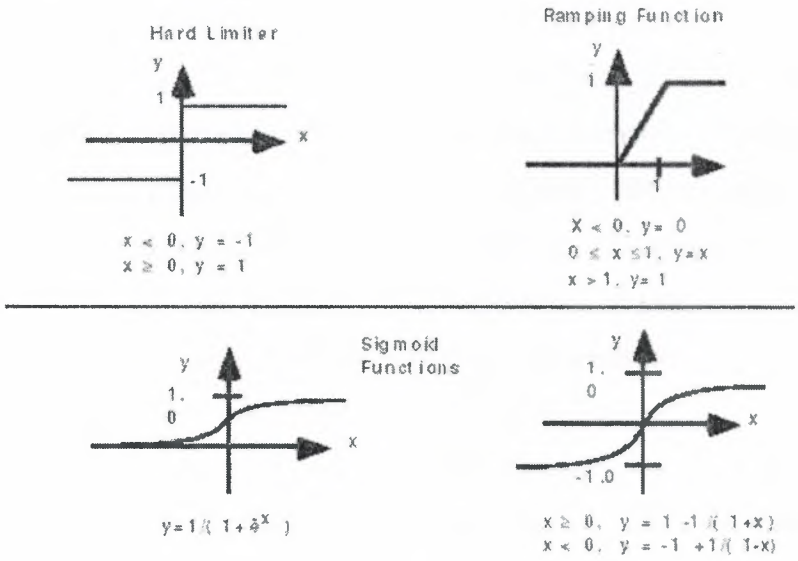


Figure 2.4 Sample Transfer Functions

functions are limited because the output is simply proportional to the input. Linear functions are not very useful. That was the problem in the earliest network models as noted in Minsky and Papert's book *Perceptrons*. The transfer function could be something as simple as depending upon whether the result of the summation function is positive or negative. The network could output zero and one, one and minus one, or other numeric combinations. The transfer function would then be a "hard limiter" or step function. See Figure 2.4 for sample transfer functions.

Another type of transfer function, the threshold or ramping function, could mirror the input within a given range and still act as a hard limiter outside that range. It is a linear function that has been clipped to minimum and maximum values, making it non-linear. Yet another option would be a sigmoid or S-shaped curve. That curve approaches a minimum and maximum value at the asymptotes. It is common for this curve to be called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent when it ranges between -1 and 1. Mathematically, the exciting feature of these curves is that both the function and its derivatives are continuous. This option works fairly well and is often the transfer function of choice. Other transfer functions are dedicated to specific network architectures. Prior to applying the transfer function, uniformly distributed random noise may be added. The source and amount of this noise is determined by the learning mode of a given network paradigm. This noise is normally referred to as "temperature" of the artificial neurons. The name, temperature, is derived from the physical phenomenon that as people become too hot or cold their ability to think is affected. Electronically, this process is simulated by adding noise. Indeed, by adding different levels of noise to the summation result, more brain-like transfer functions are realized. To more closely mimic nature's characteristics, some experimenters are using a gaussian noise source. Gaussian noise is similar to uniformly distributed noise except that the distribution of random numbers within the temperature range is along a bell curve. The use of temperature is an ongoing research area and is not being applied to many engineering applications.

NASA announced a network topology, which uses what it calls a *temperature coefficient* in a new feed-forward, back-propagation learning function. But this temperature coefficient is a global term that is applied to the gain of the transfer function. It should not be confused with the more common term, *temperature*, which is simple noise being added to individual neurons. In contrast, the global temperature coefficient allows the transfer function to have a learning variable much like the

synaptic input weights. This concept is claimed to create a network, which has a significantly faster (by several order of magnitudes) learning rate and provides more accurate results than other feed forward, back-propagation networks.

2.4.2.4 Scaling and Limiting

After the processing element's transfer function, the result can pass through additional processes, which scale and limit. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism, which insures that the scaled result does not exceed an upper, or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed. This type of scaling and limiting is mainly used in topologies to test biological neuron models, such as James Anderson's brain-state-in-the-box [1].

2.4.2.5 Output Function (Competition)

Each processing element is allowed one output signal, which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

2.4.2.6 Error Function and Back-Propagated Value

In most learning networks the difference between the current output and the desired output is calculated. This raw error is then transformed by the error function to match particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error. The current error is typically propagated backwards to a previous layer. Yet, this back-propagated value can be

either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle.

2.4.2.7 Learning Function

The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the *adaption function*, as well as the learning mode. There are two types of learning: *supervised* and *unsupervised*. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network.

2.5 Comparing Neural Networks and Traditional Computing

Neural networks offer a different way to analyze data, and to recognize patterns within that data, than traditional computing methods. However, they are not a solution for all computing problems. Traditional computing methods work well for problems that can be well characterized. Balancing checkbooks, keeping ledgers, and keeping tabs of inventory are well defined and do not require the special characteristics of neural networks. Table 2.1 identifies the basic differences between the two computing approaches. Traditional computers are ideal for many applications. They can process data, track inventories, network results, and protect equipment. These applications do not need the special characteristics of neural networks.

2.6 Network Layers

Basically, all artificial neural networks have simple topologic structures. Some neuron are used to get inputs from real world and some other neurons are used to form the real world at the output of the network. All remained neurons are called hidden neurons because of their invisibility.

Table 2.1 Comparison of Computing Approaches

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing Style	Sequential	Parallel
Functions	Logically (left brained) via Rules Concepts Calculations	Gestault (right brained) via Images Pictures Controls
Functions	Logically (left brained) via Rules Concepts Calculations	Gestault (right brained) via Images Pictures Controls
Learning Method	by rules	by example
Applications	Accounting, word processing math, inventory, digital communications	Sensor processing, speech recognition, pattern recognition, text recognition

When inputs reach to input layer, neurons produce outputs that are the inputs of other layers. The number of the hidden neurons is so important because if we use too much hidden neurons in out network, we cannot reach to desired output. And it means, there is a generalization in our network.

2.7 Communication and Types of Connections

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

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

2.7.1 Inter-Layer Connections

There are different types of connections used between layers; these connections between layers are called inter-layer connections [3].

2.7.1.1 Fully Connected

Each neuron on the first layer is connected to every neuron on the second layer (Figure 2.5) [3].

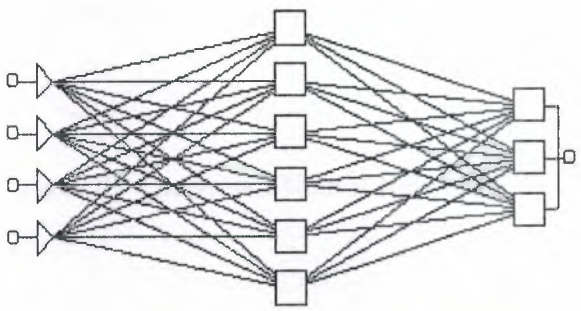


Figure 2.5 Fully Connected Neural Networks

2.7.1.2 Partially Connected

A neuron of the first layer does not have to be connected to all neurons on the second layer (Figure 2.6) [3].

2.7.1.3 Feed Forward

The neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back form the neurons on the second layer (Figure 1.7) [3].

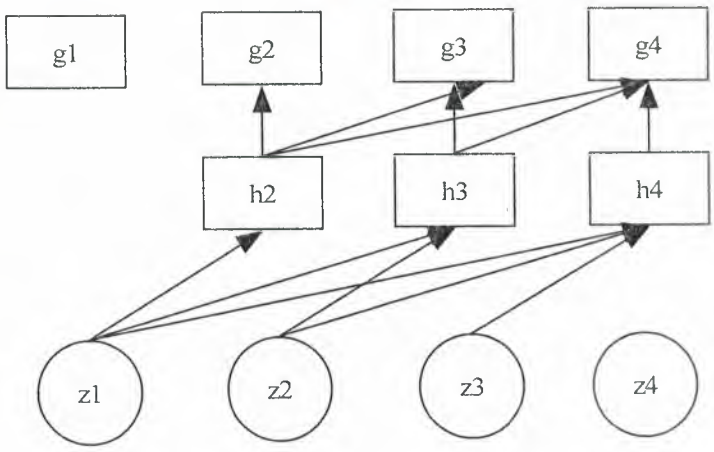


Figure 2.6 Partially Connected Neural Networks

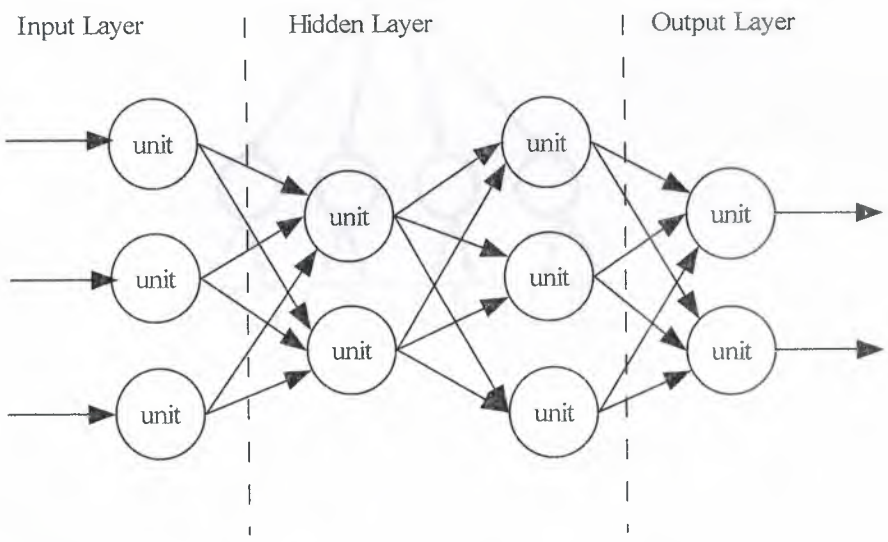


Figure 2.7 Feed Forward Neural Networks

2.7.1.4 Bi-directional

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

Feed forward and bi-directional connections could be fully- or partially connected [3].

2.7.1.5 Hierarchical

If a neural network has a hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer (figure 2.8) [3].

2.7.1.6 Resonance

The layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved [3].

2.7.2 Intra-Layer Connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections [3].

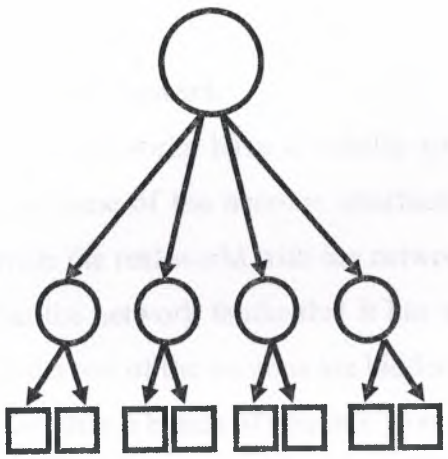


Figure 2.8 Classic Hierarchical Connection

2.7.2.1 Recurrent

The neurons within a layer are fully- or partially connected to one another. After these neurons receive input form another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer [3].

2.7.2.2 On-Center / Off-Surround

A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal

interchange, the neurons with an active output value will win, and is allowed to update its and its gang member's weights. (There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.)

2.8 A Simple Artificial Neural Network

Basically, all artificial neural networks have a similar structure or topology as shown in Figure 2.9. In that structure some of the neurons interfaces to the real world to receive its inputs. Other neurons provide the real world with the network's outputs. This output might be the particular character that the network thinks that it has scanned or the particular image it thinks is being viewed. All the rest of the neurons are hidden from view.

But a neural network is more than a bunch of neurons. Some early researchers tried to simply connect neurons in a random manner, without much success. Now, it is known that even the brains of snails are structured devices. One of the easiest ways to design a structure is to create layers of elements. It is the grouping of these neurons into layers, the connections between these layers, and the summation and transfer functions that comprises a functioning neural network. The general terms used to describe these characteristics are common to all networks.

Although there are useful networks, which contain only one layer, or even one element, most applications require networks that contain at least the three normal types of layers - input, hidden, and output. The layers of input neurons receive the data either from input files or directly from electronic sensors in real-time applications. The output layer sends information directly to the outside world, to a secondary computer process, or to other devices such as a mechanical control system. Between these two layers can be many hidden layers. These internal layers contain many of the neurons in various interconnected structures. The inputs and outputs of each of these hidden neurons simply go to other neurons.

In most networks each neuron in a hidden layer receives the signals from all of the neurons in a layer above it, typically an input layer. After a neuron performs its function it passes its output to all of the neurons in the layer below it, providing a feed forward path to the output. These lines of communication from one neuron to another are important aspects of neural networks. They are the glue to the system. They are the connections, which provide a variable strength to an input. There are two types of these connections. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. In more human terms one excites while the other inhibits.

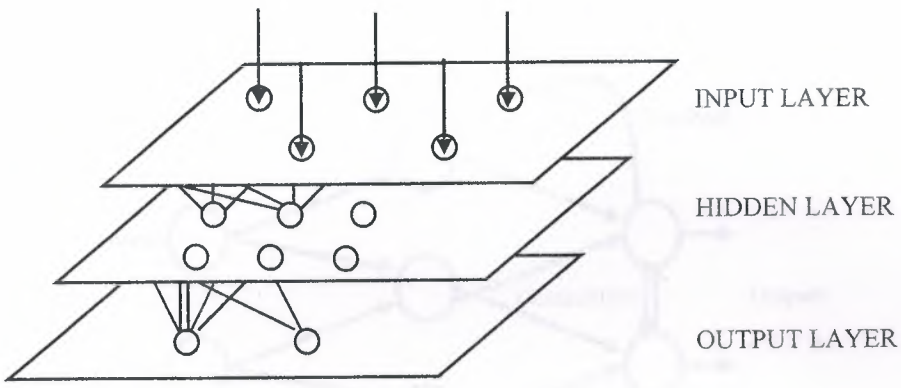


Figure 2.9 Simple Neural Network Diagram

Some networks want a neuron to inhibit the other neurons in the same layer. This is called *lateral inhibition.* The most common use of this is in the output layer. For example in text recognition if the probability of a character being a "P" is .85 and the probability of the character being an "F" is .65, the network wants to choose the highest probability and inhibit all the others. It can do that with lateral inhibition. This concept is also called *competition.*

Another type of connection is *feedback.* This is where the output of one-layer routes back to a previous layer. An example of this is shown in Figure 2.10.

The way that the neurons are connected to each other has a significant impact on the operation of the network. In the larger, more professional software development packages the user is allowed to add, delete, and control these connections at will. By "tweaking" parameters these connections can be made to either excite or inhibit.

2.9 Learning

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

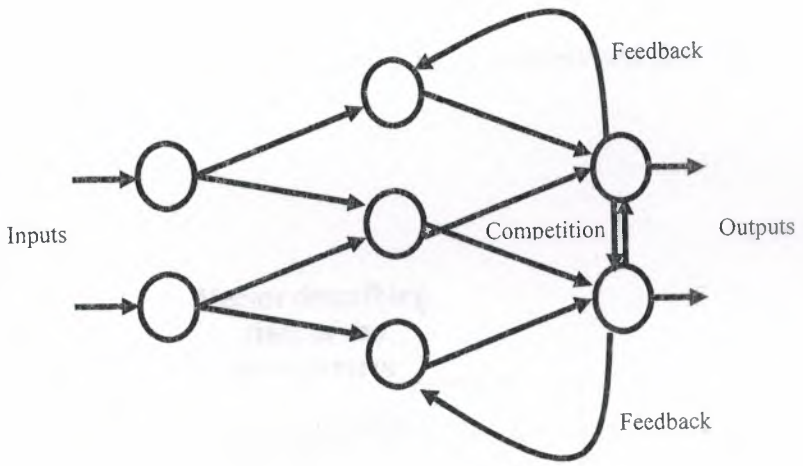


Figure 2.10 Simple Network with Feedback and Competition

The training method usually consists of one of two schemes:

2.9.1 Unsupervised learning

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing. Figure 2.11

2.9.2 Supervised Learning

In a supervised learning process , the input data and its corresponding output are presented to the neural network . The neural network , according to defined

law , change its weights in order to be able to reproduce the correct output , when an input is applied . Figure 2.12

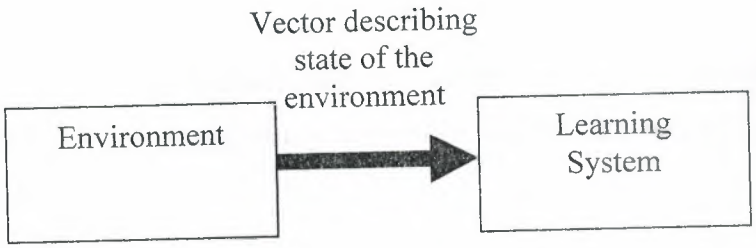


Figure 2.11 Unsupervised learning

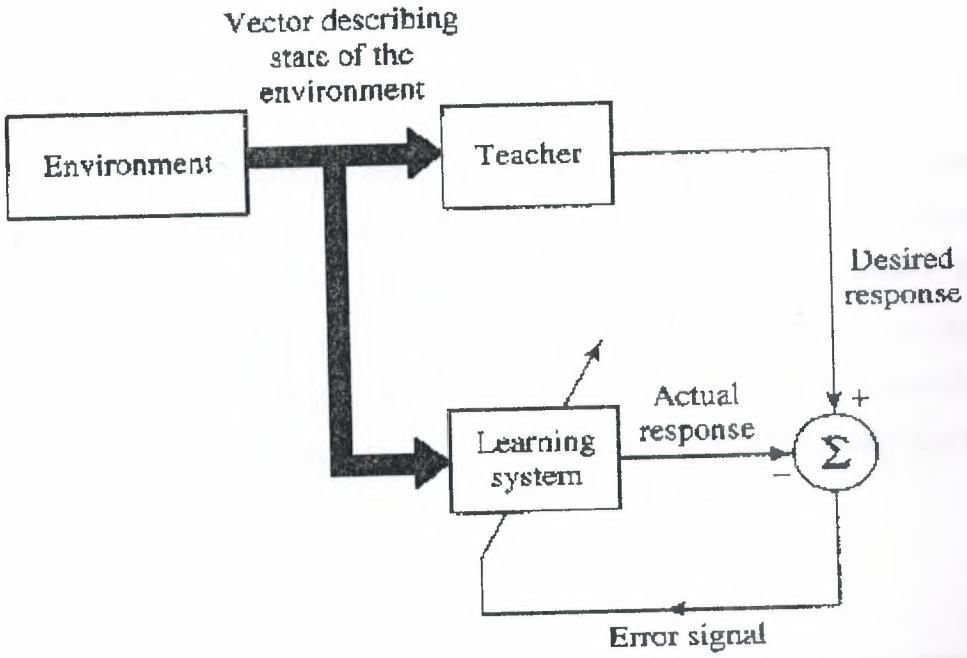


Figure2.12 Supervised learning

2.10 Off-line and On-line Learning

One can categorize the learning methods into yet another group, off-line or on-line. When the system uses input data to change its weights to learn the domain knowledge.

the system could be in training mode or learning mode. When the system is being used as a decision aid to make recommendations, it is in the operation mode; this is also sometimes called recall.

2.10.1 Off-line

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

2.10.2 On-line

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

2.11 Learning laws

There is a variety of learning laws, which are in common use [1]. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule. Man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc

2.12 Network Selection

Because all artificial neural networks are based on the concept of neurons, connections, and transfer functions, there is a similarity between the different structures, or architectures, of neural networks. The majority of the variations stems from the various learning rules and how those rules modify a network's typical topology. The following sections outline some of the most common artificial neural networks. They are organized in very rough categories of application. These categories are not meant to be exclusive, they are merely meant to separate out some of the confusion over network architectures and their best matches to specific

applications. Basically, most applications of neural networks fall into the following five categories:

- Prediction
- Classification
- Data association
- Data conceptualization
- Data filtering

Table 2.2 shows the differences between these network categories and shows which of the more common network topologies belong to which primary category. This chart is intended as a guide and is not meant to be all-inclusive. Some of these networks, which have been grouped by application, have been used to solve more than one type of problem. Feed forward back-propagation in particular has been used to solve almost all

Table 2.2 Network Selector Table[1]

NETWORK TYPE	NETWORKS	USE FOR NETWORK
Prediction	-Back Propagation -Delta Bar Delta -Extended Delta Bar Delta -Directed Random Search -Higher order Neural Networks -Self Organizing Map into Back Propagation	Use input values to predict some output (e.g. pick the best stocks in the stock market, predict the weather, identify people with cancer risk)
Classification	-Learning vector quantization -Counter propagation -Probabilistic Neural Networks	Use input values to determine the classification (e.g. is the input the letter A, is the blob of the video data a plane and what kind of plane is it)
Data Association	-Hopfield -Boltzman Machine -Hamming Network -Bi-directional associative memory -Spatio-temporal pattern recognition	Like classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but also identify when the scanner doesn't work properly)
Data Conceptualization	-Adaptive Resonance Network -Self organizing map	Analyze the input so that grouping relationships can be inferred (e.g. extract from a data base the names of those most likely to buy a particular product)
Data Filtering	-Recirculation	Smooth an input signal (e.g. take the noise out of a telephone signal)

types of problems and indeed is the most popular for the first four categories. The next five subsections describe these five network types.

2.13 Summary

In this chapter, the backgrounds of artificial neural networks and necessary information about them were explained. Now, the parts of neural networks that are used in the application part of this thesis will be focused.

CHAPTER 3
LEARNING METHODS IN NEURAL NETWORKS

3.1 Overview

The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training. In this chapter, all learning methods and their algorithms will be described in details.

The training method usually consists of one of two schemes:

- Unsupervised learning
- Supervised Learning

3.2 Learning Methods

3.2.1 Unsupervised Learning

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing.

3.2.1.1 Unsupervised Learners

a- Kohonen's Learning:

Kohonen suggested that one of the important mechanism in the human brain is placement of neurons in an orderly manner . Kohonen's

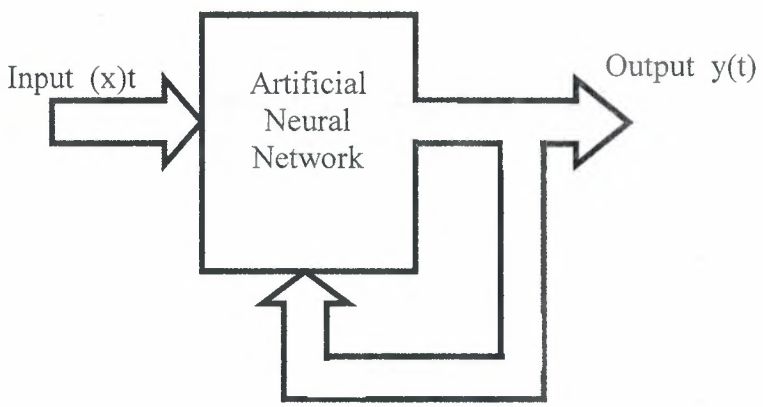


Figure 3.1 - Unsupervised Learning

learning algorithm creates a *feature map* by adjusting weights from input vectors to output vectors in a *two layer* network . The first layer is input . The second is the *competitive layer* . The two layers are fully interconnected . Input vectors are presented sequentially to layer L1 (input) . Each unit computes the dot product of its weight with the input vector . The unit with the highest dot product is declared the winner . This and its neighbors are the only units allowed to learn .

b- Competitive Learning

The simplest way to implement competitive learning is where each unit in the *hidden* or output layers receives input from all the units in the preceding layers . Within the layer units are broken down into a set of *inhibitary clusters* . The units within the clusters compete with one another to respond to data appearing at the input layer . The move strongly any particular units responds to incoming stimulus the more it inhibits other units within in the cluster . The unit learns by shifting a fraction of its weights from its inactive lines. The main disadvantage of competitive learning in the loss of previous learnings (Figure 3.3).

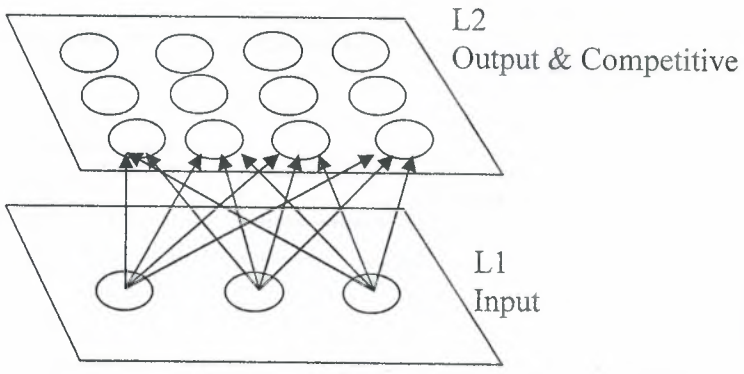


Figure 3.2 Kohonen's Learning

c- Adaptive Resonance Theory (ART)

Divided into 2 methods :

- Accept only binary
- Accept binary & continuous input

3.2.2 Supervised Learning

In a supervised learning process , the input data and its corresponding output are presented to the neural network . The neural network , according to defined law , change its weights in order to be able to reproduce the correct output , when an input is applied .

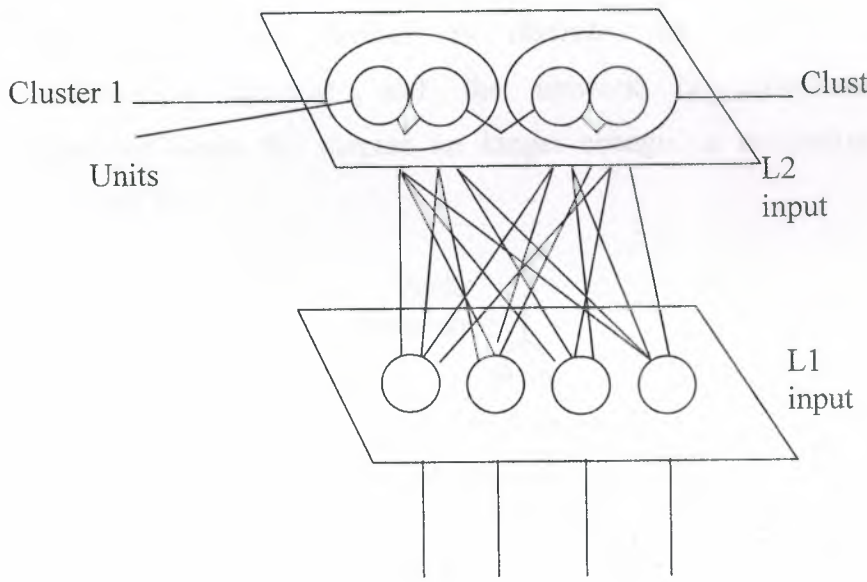


Figure 3.3 Competitive Learning

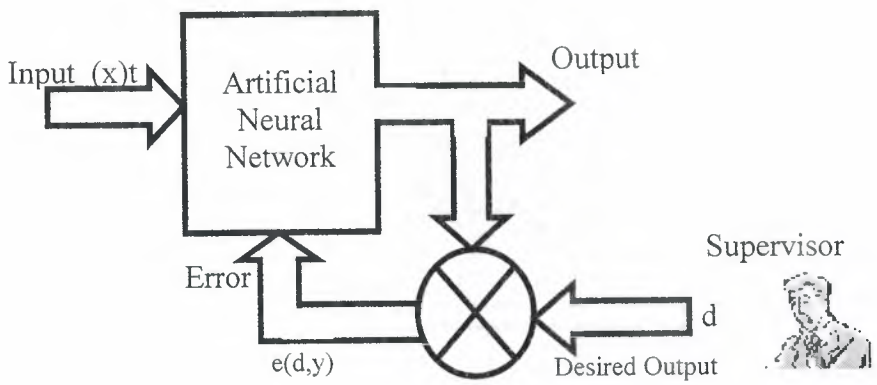


Figure 3.4 - Supervised Learning

3.2.2.1 Supervised Learners

a- The Perceptron

This can be trained and can make decisions . During the training phase, *pairs* of input & output vectors are used to train the network . With each input vector , the output vector is compared with a *desired* output *(target)* and the error between the *actual* and the *desired* output vectors is used to update the weights.

b- Hopfield Network

It is essentially used with binary number . Weights are initialized using training samples . In the decision making phase , the test data is presented to the net at certain time . Following initialization the Hopfield Network *iterates in discrete time* stops using some mathematical function , and the network is considered to have converged when the outputs no longer change on successive iterations (Figure 3.7) .

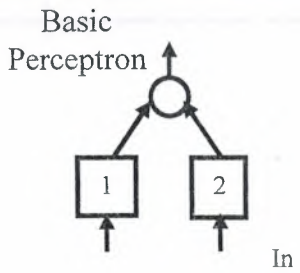


Figure 3.5 Basic Perceptron

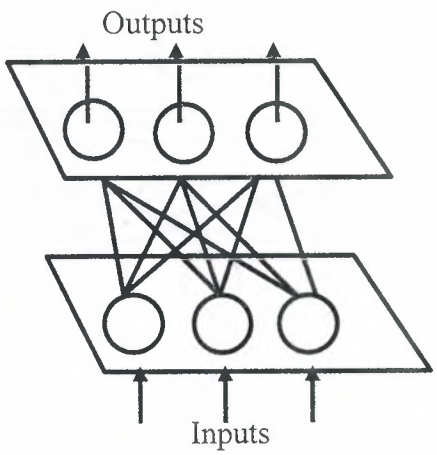


Figure 3.6 The Perceptron Model

c- Hamming Network

It is similar to Hopfield network . As shown in the figure 2.8, it consists of four layer.

L1 : Input layer

L2 : Calculates matching scores

L3 : Feedbacks as in Hopfield

L4 : Output layer

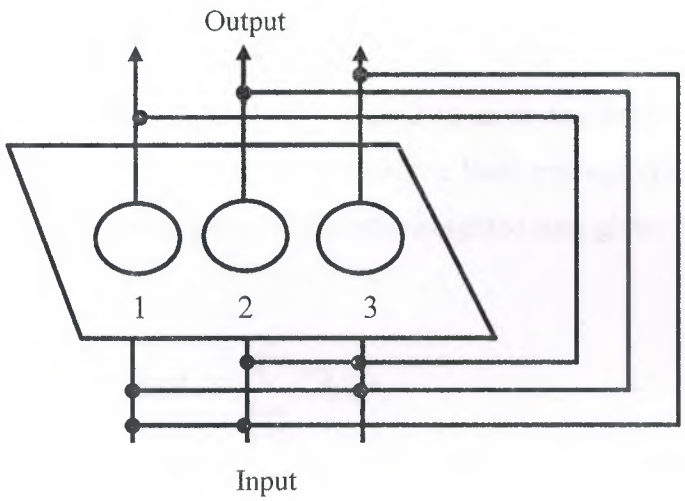


Figure 3.7 Hopfield Network

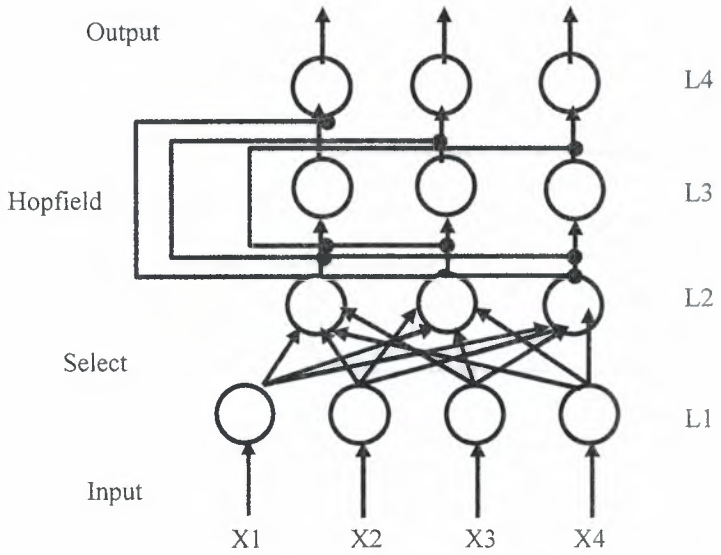


Figure 3.8 Hamming Network

d- Back Propagation Training Algorithm

The equations that describe the network training and operation can be divided into two categories . First, the *feed-forward* calculations. These are used in both training mode in the operation of the trained neural network. Second, the *error back propagation* calculations. These are applied only during training. But before we present the two categories of calculations, we have to describe another important element, *activation function* that the algorithm will be based upon.

i - The Activation Function

An artificial neuron (Figure 2.1), as it was described in chapter 1, is the fundamental building block in a back propagation network. The input to the neuron is obtained as the weighted sum given by equation (2.1).

$$net = \sum_{i=1}^{n} O_i w_i \qquad (2.1)$$

In figure 2.9, F is the activation function, which has a sigmoid form. The simplicity of the derivative of the sigmoid function justifies it's popularity and use as an activation function in training algorithms. With a sigmoid activation function the output of the neuron is given by equation (2.2) and (2.3).

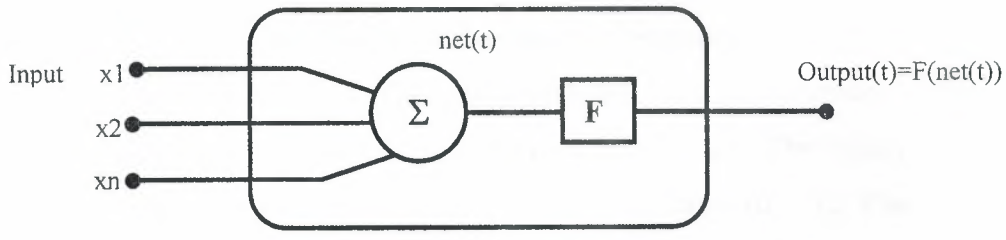


Figure 3.9 Artificial Neuron

$$out = F(net) \qquad (2.2)$$

$$F(net) = \frac{1}{(1 + \exp(-net))} \tag{2.3}$$

The derivative of the sigmoid function can be obtained as follows:

$$\frac{\partial F(net)}{\partial net} = \frac{\exp(-net)}{(1 + \exp(-net))^2}$$

$$= \left(\frac{1}{1 + \exp(-net)}\right)\left(\frac{\exp(-net)}{1 + \exp(-net)}\right)$$

$$= out(1 - out)$$

$$= F(net)[1 - F(net)] \tag{2.4}$$

Any other function that is differentiable everywhere can be used in the back propagation algorithm. For example, linear functions with adjustable gain, relay functions with threshold characteristics, linear threshold characteristic functions and Sigmoid functions for different values of gain, are all common activation functions that can be used.

ii- Feed Forward Calculations

Figure 2.10 shows the most common configuration of a back propagation neural network. This is the simple three layer back propagation model. Each neuron is represented by a circle and each interconnection, with its associated weight, by an arrow. The neurons labeled *b* are bias neurons. Normalization of the input data prior to training is necessary. The values of the input data into the input layer must be in the range (0 – 1). The stages of the feed forward calculations can be described according to the layers. The suffixes *i*, *h*, and *j* are used for *input*, *hidden* and *output* respectively.

ii.1 Input Layer (*i*)

Figure 3.11 shows a neuron in the input layer. The output of each input layer neuron is exactly equal to the normalized input.

$$Input - Layer \;\; Output = O_i = I_i \tag{3.5}$$

ii.2 Hidden Layer (*h*)

Figure 3.12 describes a neuron in the hidden layer. The signal presented to a neuron in the hidden layer is equal to the sum of all outputs of the input layer neurons multiplied by their associated connection weights, as in equation (2.6).

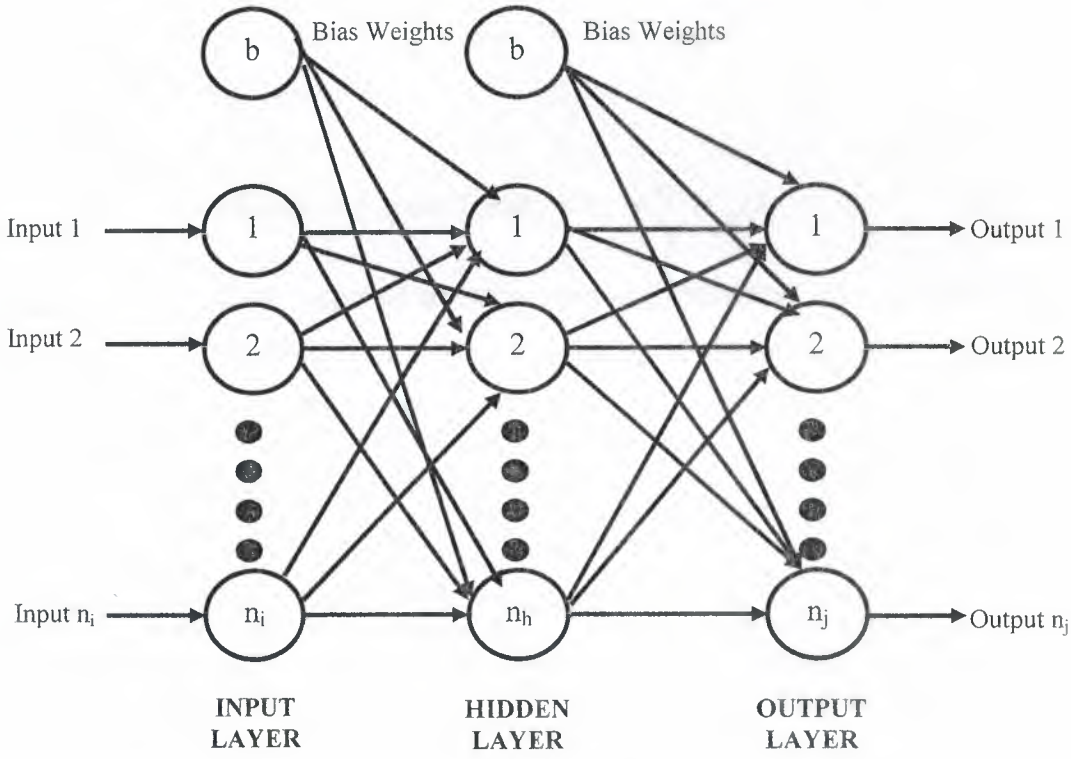


Figure 3.10 Back Propagation Network Structure

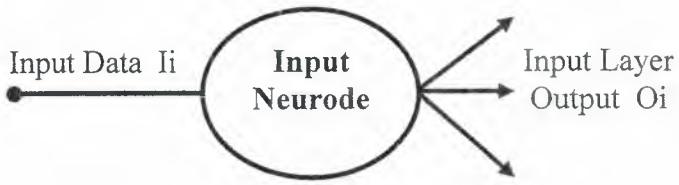


Figure 3.11 An Input Layer Neuron

$$Hidden-Layer\ Input_h = I_h = \sum_i W_{hi} O_i \qquad (3.6)$$

Each output of a hidden neuron is calculated using the sigmoid function. This is described in equation (3.7).

$$Hidden-Layer\ Output_h = O_h = \frac{1}{1+\exp(-I_h)} \qquad (3.7)$$

ii.3 Output Layer (*j*)

Figure 3.13 describes a neuron in the output layer. The signal presented to a neuron in the output layer is equal to the sum of all outputs of the hidden layer neurons multiplied by their associated weights plus the bias weights at each neuron, as in equation (3.8).

$$Output-Layer\ Input_j = I_j = \sum_h W_{jh} O_h \qquad (3.8)$$

Each output of an output neuron is calculated using the sigmoid function in a similar manner as in the hidden layer. This is described in equation (3.9).

$$Output-Layer\ Output_j = O_j = \frac{1}{1+\exp(-I_j)} \qquad (3.9)$$

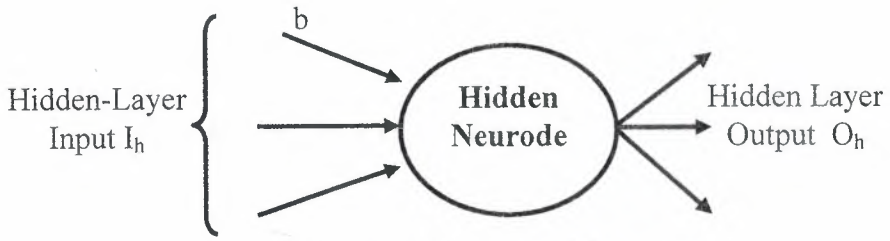


Figure 2.12 A Hidden Layer Neuron

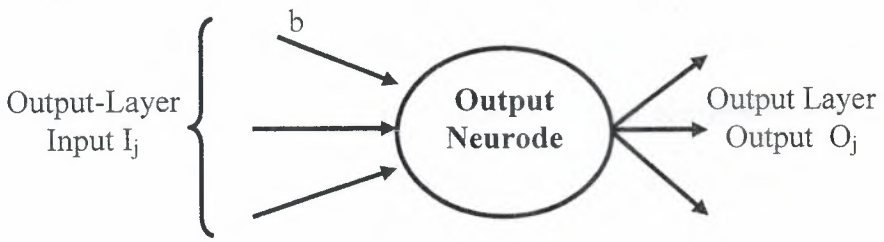


Figure 3.13 An Output Layer Neuron

The set of calculations that has been described so far in the feed forward calculations, can be carried out during the training phase as well as during the testing / running phase.

iii- Error Back Propagation Calculations

The error back propagation calculations are applied only during the training of the neural network. Vital elements in these calculations are described below. These include, the error signal, some essential parameters and weight adjustment.

iii.1 Signal Error

During the network training, the feed forward output state calculation is combined with backward error propagation and weight adjustment calculations that represents the network's learning. Central to the concept of training a neural network is the definition of network *error*. Rumelhart and McClelland define an error term that depends on the difference between the output neuron is supposed to have, called the target value T_j, and the value it actually has as a result of the feed forward calculations, O_j. The error term represents a measure of how well a network is training on a particular training set.

Equation (10) presents the definitions for the error. The subscript p denotes what the value is for a given pattern.

$$E_p = \sum_{j=1}^{n_j} (T_{pj} - O_{pj})^2 \qquad (2.10)$$

The aim of the training process is to minimize this error over all training patterns. From equation (2.9), it can be seen that the output of a neuron in the output layer is a function of its input, or $O_j = f(I_j)$. The first derivative of this function, $f'(I_j)$ is an important element in error back propagation. For output layer neurons, a quantity called the error signal

is represented by Δ_j, which is defined in equation (2.11) and thus equation (2.12).

$$\Delta_j = f'(I_j)(T_j - O_j) \tag{2.11}$$

$$= (T_j - O_j)O_j(1 - O_j) \tag{2.12}$$

This error value is propagated back and appropriate weight adjustments are performed. This is done by accumulating the Δ's for each neuron for the entire training set, add them, and propagate back the error based on the grand total Δ. This called *batch (epoch) training.*

iii.2 Essential Parameters

There are two essential parameters that do affect the learning capability of the neural network. First, the *learning coefficient* η that defines the learning 'power' of a neural network. Second, the *momentum factor* α, which defines the speed at which, the neural network learns. This can be adjusted to a certain value in order to prevent the neural network from getting caught in what is called *local energy minima.* Both rates can have a value between 0 and 1.

iii.3 Weight Adjustment

Each weight has to be set to an initial value. Random initialization is usually performed. Weight adjustment is performed in stages. Starting at the end of the feed forward phase, and going backward to the inputs of the hidden layer.

iii.3.a Output-Layer Weights Update

The weights that feed the output layer (W_{jh}) are updated using equation (2.13). This also includes the bias weights at the output layer neurons. However, in order to avoid the risk of the neural

network getting caught in local minima, the momentum term can be added as in equation (14).

$$W_{jh}(new) = W_{jh}(old) + \eta \Delta_j O_h \qquad (2.13)$$

$$W_{jh}(new) = W_{jh}(old) + \eta \Delta_j O_h + \alpha \left[\delta W_{jh}(old) \right] \qquad (2.14)$$

Where $\delta W_{jh}(old)$ stands for the previous weight change.

iii.3.b Hidden-Layer Weights Update

The error term for an output layer is defined in equation (2.12). For the hidden layer, it is not as simple to figure out a definition for the error term. However, a definition by Rumelhart and McClelland describes the error term for a hidden neuron as in equation (2.15) and, subsequently, in equation (2.16).

$$\Delta_h = f'(I_h) \sum_{j=0}^{n_j} W_{jh} \Delta_j \qquad (2.15)$$

$$\Delta_h = O_h (1 - O_h) \sum_{j=0}^{n_j} W_{jh} \Delta_j \qquad (2.16)$$

The weight adjustments for the connections feeding the hidden layer from the input layer are now calculated in a similar manner to those feeding the output layer. These adjustments are calculated using equation (2.17).

$$W_{hi}(new) = W_{hi}(old) + \eta \Delta_h O_i + \alpha \left[\delta W_{hi}(old) \right] \qquad (2.17)$$

The bias weights at the hidden layer neurons are updated, similarly, using equation (2.17).

3.3 Summary

In this chapter, the learning methods of neural networks and their learners were presented. Also supervised and Unsupervised Learning methods were described in details.

In Unsupervised Learning, Kohonen's Learning, Competitive Learning and Adaptive Resonance Theory were described.

In Supervised Learning, A Perceptron, Hopfield, Hamming and Back Propagation Learning were described.

CHAPTER 4
CHARACTER RECOGNITION SYSTEM & RESULTS

4.1 Overview

This chapter presents developed character recognition system in details. Final parameters, used methods and character databases are presented.

4.2 Character Recognition System

Developed Character Recognition System (CRS) has been implemented using Matlab program. It consists two phases: Training Phase and Testing Phase.

4.2.1 Training Phase of CRS

26 English characters are used to train neural network. As shown in the network topology (Figure 4.1), neural network has 2500 inputs, 1 hidden layer with 30 neurons, and 26 output neurons. In this part, both training and training phases of neural networks will be described.

4.2.1.1 Inputs of Neural Network

Neural Network have 2500 input neurons comes from 50x50 characters

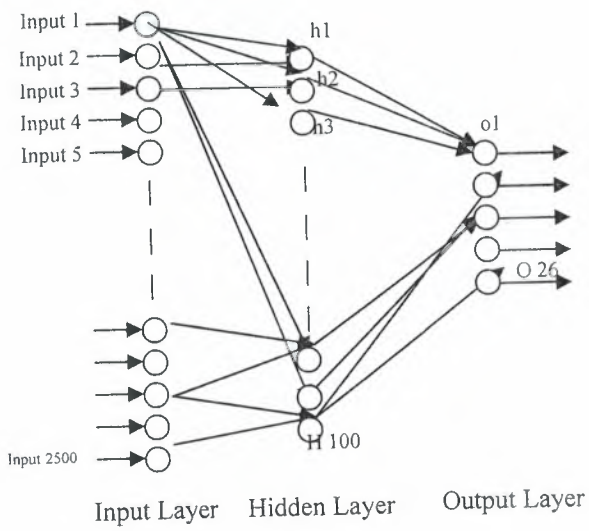


Figure 4.1 – Neural Network Topology

4.2.1.2 Hidden Layer of Network

When outputs of input layer are reached to hidden layer, it starts to calculate hidden layer inputs for each pattern using hidden layer weights. After calculating hidden layer inputs, hidden layer outputs are calculated using Sigmoid Function. After various experiments 100 hidden neurons are determined to use in system.

4.2.1.3 Output Layer of Network

Hidden layer and output layer calculations and principles are similar to each other. When outputs of hidden layer are reached to output layer, it starts to calculate output layer inputs for each pattern using output layer weights. After calculating output layer inputs, output layer outputs are calculated using Sigmoid Function.

4.2.1.4 Errors and Back Propagation

After calculating the outputs of output layer for each pattern, it is necessary to compare them with desired outputs. After comparing output layer output of each pattern with desired output, error for each pattern and error is calculated. Error of each pattern is used to update weights and Error is used for stopping condition. Error Level was assigned to 0.01 because this value is sufficient for the required accuracy of the developed system and to keep neural network training time as low as possible. When error is reached to desired level, program will save final weights and stop.

These steps will repeat until error is less than 0.01 or iterations reach to 10,000. All steps of training can be shown in figure 4.2.

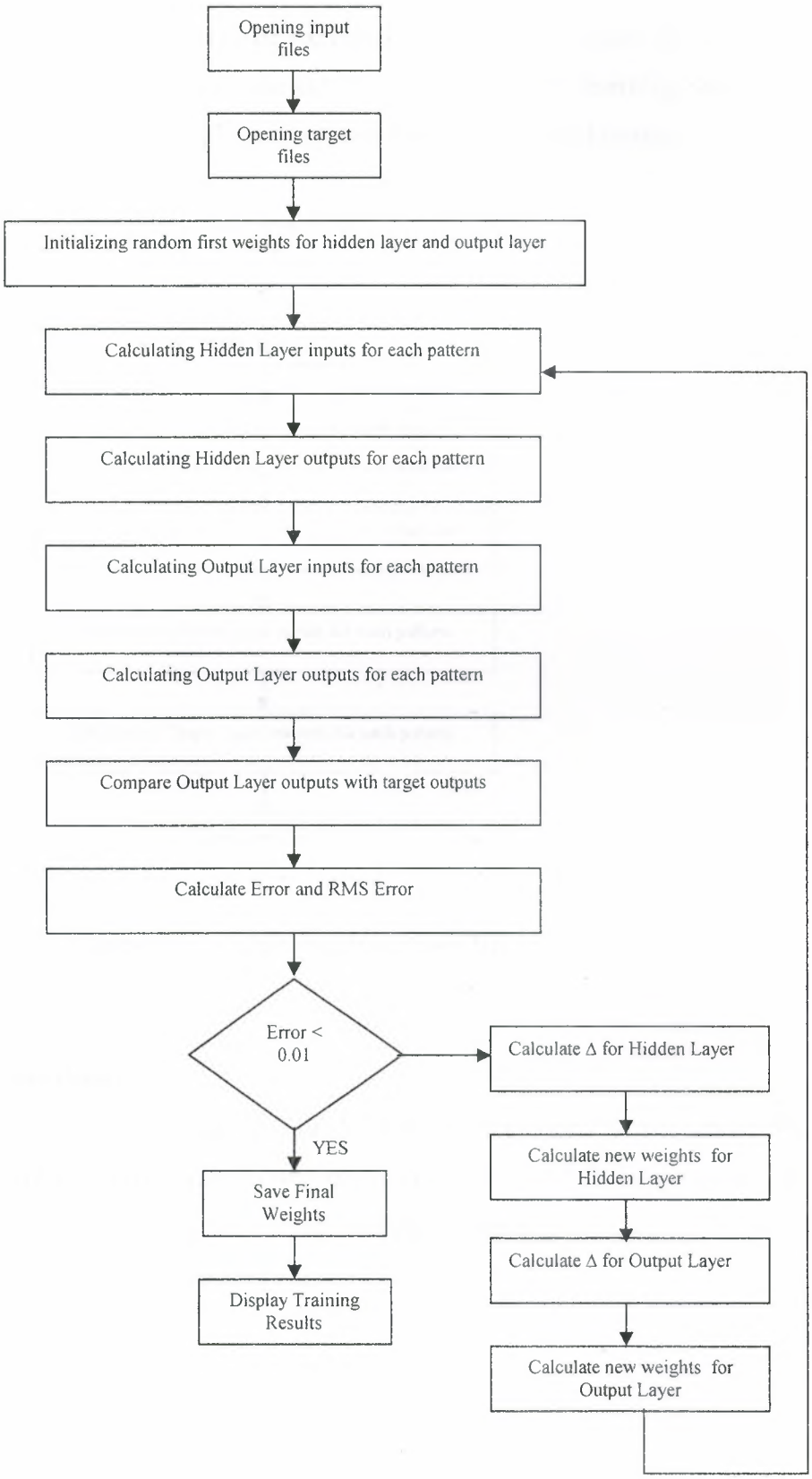


Figure 4.2 – Flowchart of CRS

4.2.2 System Testing

In running phase, only one iteration could be executed. Running phase uses the saved final weights of training program and it doesn't use any term of momentum, learning rate or error (Figure 4.3). Running phases is the last phase of system that gives us final results.

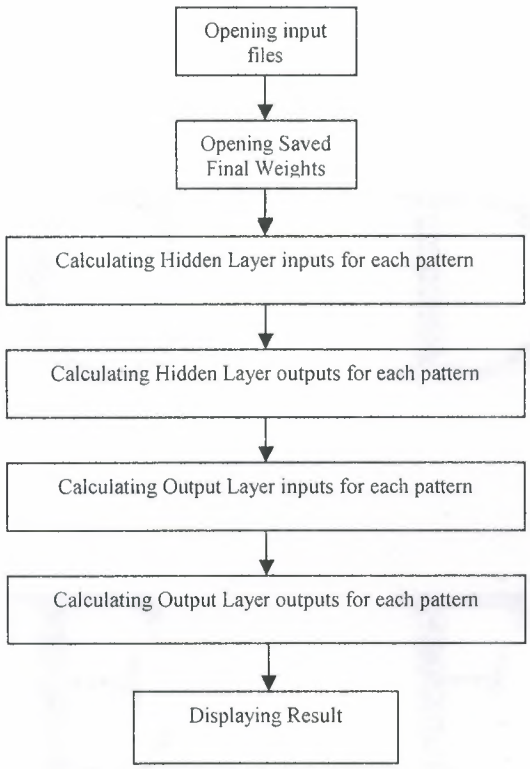


Figure 4.3 - Running Phase of CRS

4.3 Character Image Database

There are 26 characters in English alphabets and CRS are developed to recognize these characters with optimum recognition rate in minimum time. There are 2 databases in CRS. One for training database and the other for testing database. Training database can be seen in Figure 4.4 and testing database can be seen in Figure 4.5.


A B
C D
E F
G H

I J

K L

M N

O P

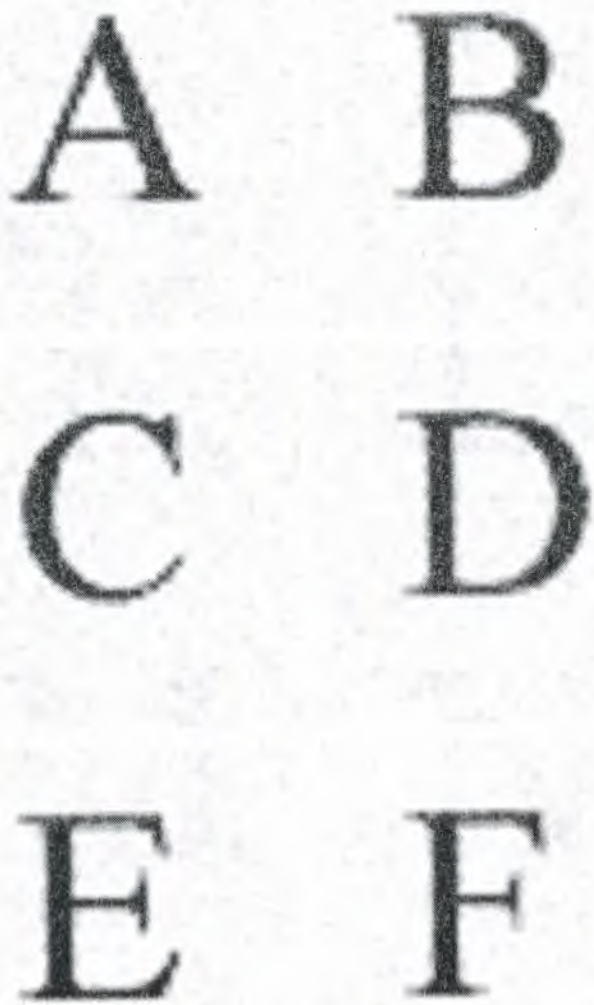
Q R

S T

U V

W X

Figure 4.4 – Training Database

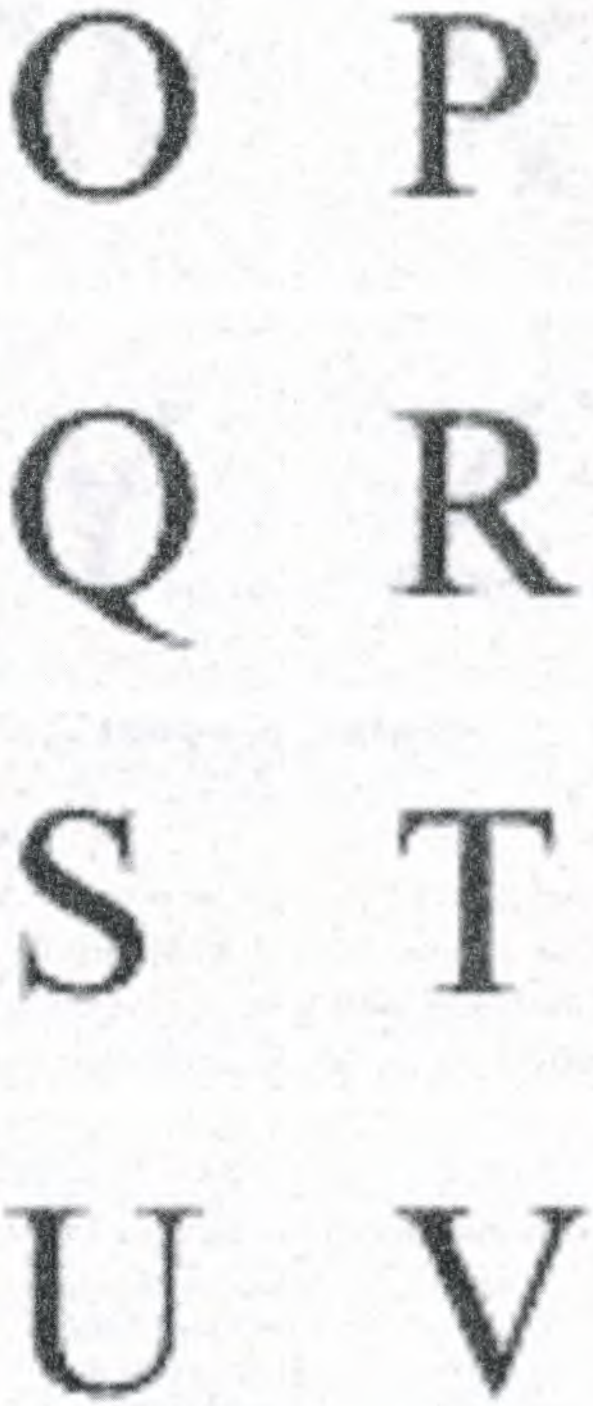


G H

I J

K L

M N


O P
Q R
S T
U V

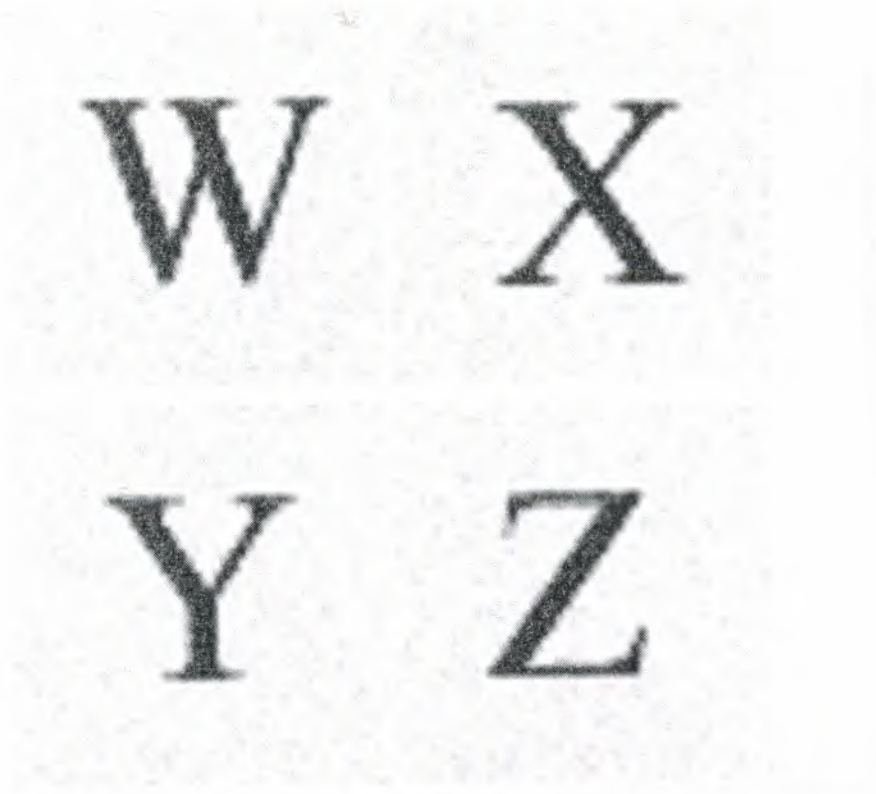


Figure 4.5 – Testing Database

4.4 Experimental Results

Experiments had been performed in two stages. In first stage, trained characters had been generalized to test the learning efficiency of the system. In second stage, noise added characters which are not trained to neural network were tested. Final neural network parameters and error-level graph of training can be seen in Table 4.1 and Figure 4.6 respectively.

Table 4.1 – Final Neural Network Parameters

Input Layer Nodes	2500
Hidden Layer Nodes	100
Output Layer Nodes	26
Learning Rate	0.003
Momentum Rate	0.15
Minimum Error	0.01
Iterations	2737
Training Time	122 sec.

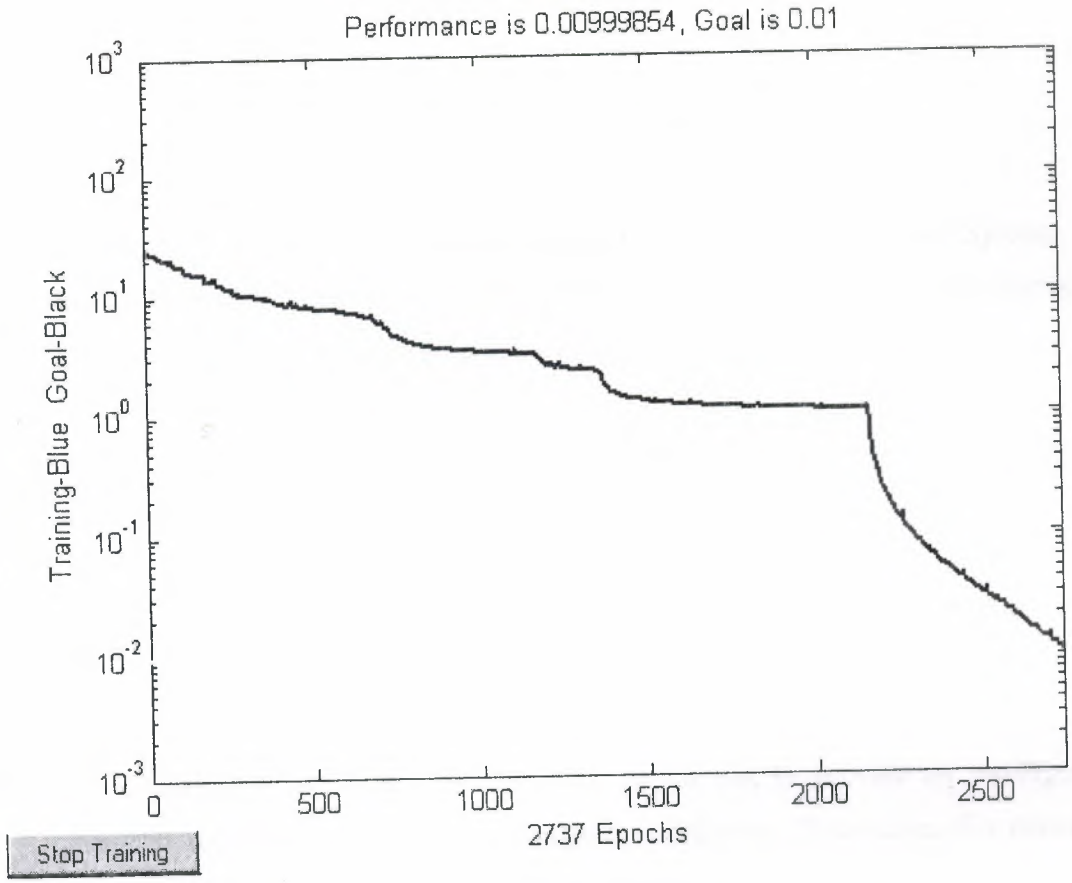


Figure 4.6 – Error Level Graph of Neural Network during Training

In stage 1, recognition rate of trained characters was reached to 100% which was expected from neural networks. In second stage, system recognized 25/26 characters which mean 96.12% of recognition rate. Totally system reached to 98.07% of success rate by recognizing 51 characters of 52. Table 4.2 shows the general results of experiments.

Table 4.2 – Experimental Results

Trained Patterns	26/26 100%
Noisy Patterns	25/26 96.12%
Total	51/52 98.07%

4.5 Overview

In this chapter, all details of CRS and performed experiments were presented in details.

CONCLUSION

Project is devoted one of the actual problem Character Recognition System. A set of English characters have been applied to the NN recognition system. These characters were recognized by using the back-propagation NN.

Moreover, to confirm the recognition ability and flexibility of the system, noisy character images were applied to the system.

From the experimental results, it seems that this recognition system is effective for clean and noisy character images of Times New Roman font.

The investigation of Neural Networks is carried out to provide an intelligent system Character Recognition System that recognizes characters. Simulation of a neural network for character recognition using Matlab, carried out.

REFERENCES

[1]http://pocket-pc-software.penreader.com

[2]http://homepages.cwi.nl

[3]http://csdl2.computer.org

[4]http://www.dontveter.com

[5]http://www.dontveter.com

[6]Boran Şekeroğlu Intelligent Bnknote Identification System(IBIS) Master Thesis Nicosia-2004

APPENDIX

Program to Read Character Images

```
clc;
close all;
start_time = cputime;

a = double(rgb2gray(imread('a.jpg')));
b = double(rgb2gray(imread('b.jpg')));
c = double(rgb2gray(imread('c.jpg')));
d = double(rgb2gray(imread('d.jpg')));
e = double(rgb2gray(imread('e.jpg')));
f = double(rgb2gray(imread('f.jpg')));
g = double(rgb2gray(imread('g.jpg')));
h = double(rgb2gray(imread('h.jpg')));
i = double(rgb2gray(imread('i.jpg')));
j = double(rgb2gray(imread('j.jpg')));
k = double(rgb2gray(imread('k.jpg')));
l = double(rgb2gray(imread('l.jpg')));
m = double(rgb2gray(imread('m.jpg')));
n = double(rgb2gray(imread('n.jpg')));
o = double(rgb2gray(imread('o.jpg')));
p = double(rgb2gray(imread('p.jpg')));
q = double(rgb2gray(imread('q.jpg')));
r = double(rgb2gray(imread('r.jpg')));
s = double(rgb2gray(imread('s.jpg')));
t = double(rgb2gray(imread('t.jpg')));
u = double(rgb2gray(imread('u.jpg')));
v = double(rgb2gray(imread('v.jpg')));
w = double(rgb2gray(imread('w.jpg')));
x = double(rgb2gray(imread('x.jpg')));
y = double(rgb2gray(imread('y.jpg')));
z = double(rgb2gray(imread('z.jpg')));
```

```
Ps {1,1}= a;
Ps {1,2}= b;
Ps {1,3}= c;
Ps {1,4}= d;
Ps {1,5}= e;
Ps {1,6}= f;
Ps {1,7}= g;
Ps {1,8}= h;
Ps {1,9}= i;
Ps {1,10}= j;
Ps {1,11}= k;
Ps {1,12}= l;
Ps {1,13}= m;
Ps {1,14}= n;
Ps {1,15}= o;
Ps {1,16}= p;
Ps {1,17}= q;
Ps {1,18}= r;
Ps {1,19}= s;
Ps {1,20}= t;
Ps {1,21}= u;
Ps {1,22}= v;
Ps {1,23}= w;
Ps {1,24}= x;
Ps {1,25}= y;
Ps {1,26}= z;

l=1;
m=1;
for i = 1:26
   for j = 1:50
      for k = 1:50
         P(l,m) = Ps{1,i}(j,k) / 255;
```

```
        m = m + 1;
      end
    end
    l = l + 1;
    m = 1;
end
save('P.mat','P');
```

Training Program

```
P1=0;
% T=eye([26 2500]);
% T(1,:) = P(1,:);
% S1 = 10; S2 = 26;
% P1 = P(1,:);
% P1 = P';
% net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
%
% net.performFcn = 'sse';
% net.trainParam.goal = 0.1;
% net.trainParam.show = 20;
% net.trainParam.epochs = 1000;
% net.trainParam.mc = 0.95;
% [net,tr] = train(net,P,T);



%
% figure;
% imagesc(k);
% colormap(gray);
% title('Downsampled by Factor of 2');
% axis image;
% k
%
%
% % get stop time
% stop_time = cputime;
%
% % obtain the execution time of the program
% execution_time = stop_time - start_time;
%
% % display the execution time of the program
```

```
% disp('Execution Time: ');
% disp('--------------');
% execution_time
%
% I1 = 50;
% load('targets.dat');
% T=targets'
% k = double(k);
% for i=1:50
%     for j=1:50
%         km1((i-1)*I1+j)=k(i,j);
% % %         resim(I1)=k(i,j)/255;
% % %         I1 = I1 + 1;
%     end
% end
% %resim = double(resim);
% % km1 = km1';%resim;
% [R,Q] = size(km1);
% [S2,Q] = size(T);
% S1=10;
% P=km1';
% net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
% net.performFcn = 'sse';
% net.trainParam.lr = 0.03;

%net.trainParam.goal = 0.1;
% net.trainParam.show = 20;
% net.trainParam.epochs = 500;
% net.trainParam.mc = 0.95;
%
% [net,tr,Y,E] = train(net,P,T);

% clear;
% clc;
```

```
% close all;
%load('P.mat');

P1 = 0;

T=eye([26 26]);
% T(1,:) = P(1,:);
S1 = 10; %S2 = 26;
 for i=1:2500
    P1(1,i) = P(12,i); % Character number comes here
 end
% P1 = P(1,:);
% P1 = P;
P2 = P';
% P1 = P1';
% P = [0 1 2 3 4 5 6 7 8 9 10];
% T = [0 1 2 3 4 3 2 1 2 3 4];
S1 = 100;
[R,Q] = size(P2);
[S2,Q] = size(T);
net = newff(minmax(P2),[S1 S2],{'logsig' 'logsig'},'traingdx');

net.performFcn = 'sse';
net.trainParam.goal = 0.01;
net.trainParam.show = 20;
net.trainParam.lr = 0.003;
net.trainParam.epochs = 10000;
net.trainParam.mc = 0.15;
[net,tr] = train(net,P2,T);
save('TrainedNetwork.mat','P2','P1','T','net','tr');
```

Test Program

```
% pause
% resim = P1;
% A2 = sim(net,P1')
% A2 = compet(A2);
% answer = find(compet(A2) == 1);
% plotchar(P2(:,answer));

% figure ; imshow(a)



% clear;
% clc;
% close all;
%load('P.mat');

P1 = 0;

T=eye([26 26]);
% T(1,:) = P(1,:);
S1 = 10; %S2 = 26;
 for i=1:2500
    P1(1,i) = P(12,i); % Character number comes here
 end
% P1 = P(1,:);
% P1 = P;
P2 = P';
% P1 = P1';
% P = [0 1 2 3 4 5 6 7 8 9 10];
% T = [0 1 2 3 4 3 2 1 2 3 4];
S1 = 100;
[R,Q] = size(P2);
[S2,Q] = size(T);
```

```
net = newff(minmax(P2),[S1 S2],{'logsig' 'logsig'},'traingdx');

net.performFcn = 'sse';
net.trainParam.goal = 0.01;
net.trainParam.show = 20;
net.trainParam.lr = 0.003;
net.trainParam.epochs = 10000;
net.trainParam.mc = 0.15;
[net,tr] = train(net,P2,T);
save('TrainedNetwork.mat','P2','P1','T','net','tr');


% pause
% resim = P1;
% A2 = sim(net,P1')
% A2 = compet(A2);
% answer = find(compet(A2) == 1);
% plotchar(P2(:,answer));

% figure ; imshow(a)



% load('TrainedNetwork.mat');
resim = P1;
A2 = sim(net,P1')
A2 = compet(A2);
answer = find(compet(A2) == 1);
plotchar(P2(:,answer));

j = 1;
max = 0;
for i = 1 : 26
```

```
  if A2(i)> max
     max = A2(i);
     j = i;
  end
end

if j == 1
  a = imread('a.jpg')
  figure; imshow(a)
end

if j == 2
  b = imread('b.jpg')
  figure; imshow(b)
end

if j == 3
  c = imread ('c.jpg')
  figure; imshow(c)
end

if j == 4
  d = imread('d.jpg')
  figure; imshow(d)
end

if j == 5
  e = imread('e.jpg')
  figure; imshow(e)
end

if j == 6
  f = imread('f.jpg')
  figure; imshow(f)
```

```
end

if j == 7
   g= mread('g.jpg')
   figure; imshow(g)
end

if j == 8
   h = imread('h.jpg')
   figure; imshow(h)
end

if j == 9
   i = imread('i.jpg')
   figure; imshow(i)
end

if j==10
   j=imread('j.jpg')
   figure; imshow(j)
end

if j==11
   k=imread('k.jpg')
   figure; imshow(k)
end

if j == 12
   l = imread('l.jpg')
   figure; imshow(l)
end

if j == 13
   m= imread('m.jpg')
```

```
    figure; imshow(m)
end

if j == 14
    n = imread('n.jpg')
    figure; imshow(n)
end

if j == 15
    o = imread('0.jpg')
    figure; imshow(0)
end

if j == 16
    p = imread('p.jpg')
    figure; imshow(p)
end

if j == 17
    q = imread('q.jpg')
    figure; imshow(q)
end

if j == 18
    r = imread('r.jpg')
    figure; imshow(r)
end

if j == 19
    s = imread('s.jpg')
    figure; imshow(s)
end

if j == 20
```

```
    t = imread('t.jpg')
    figure; imshow(t)
end

if j == 21
    u = imread('u.jpg')
    figure; imshow(u)
end

if j == 22
    v = imread('v.jpg')
    figure; imshow(v)
end

if j == 23
    w = imread('w.jpg')
    figure; imshow(w)
end

if j == 24
    x = imread('x.jpg')
    figure; imshow(x)
end

if j == 25
    y = imread('y.jpg')
    figure; imshow(y)
end

if j == 26
    z = imread('z.jpg')
    figure; imshow(z)
end
```